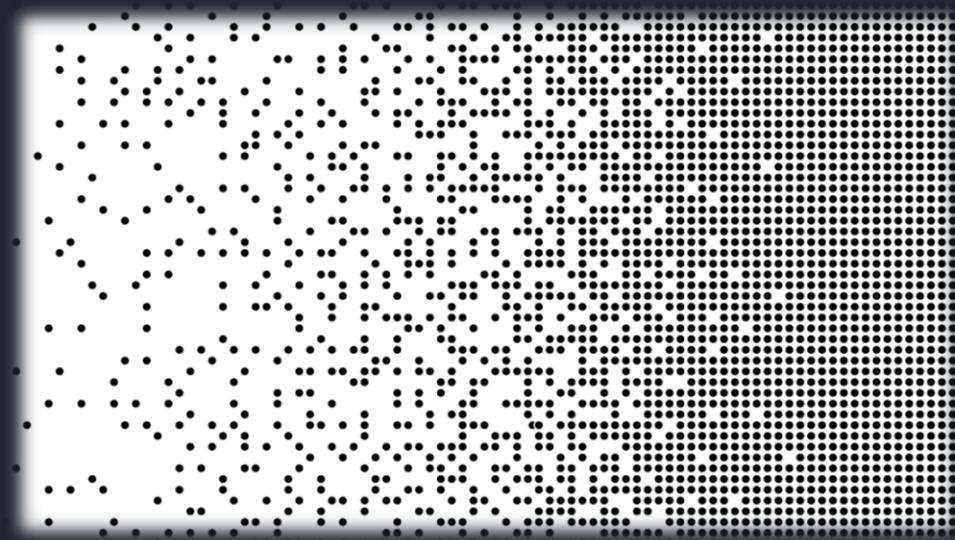


# Design Patterns



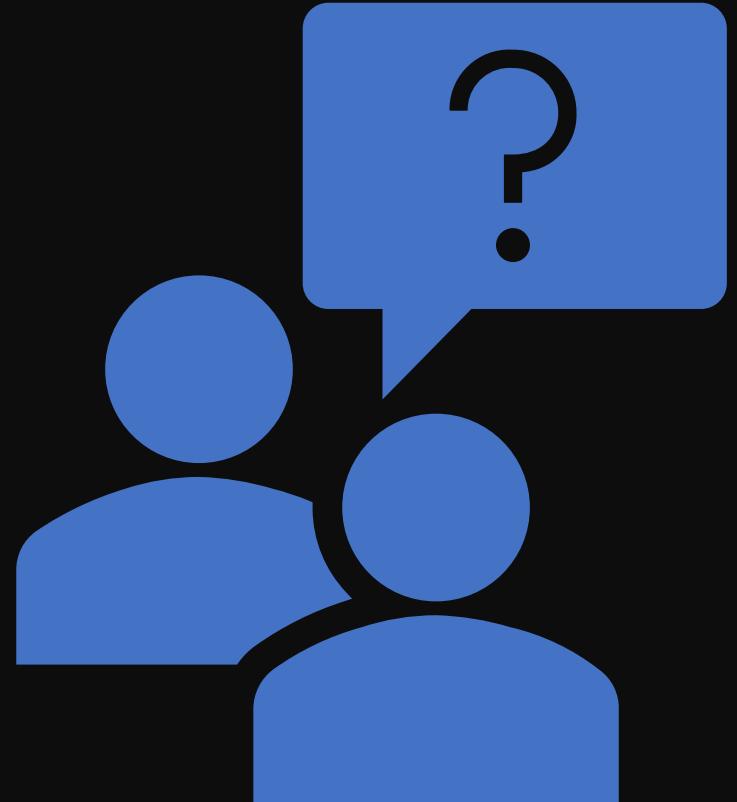
**PÓS-GRADUAÇÃO LATO SENSU  
EM ARQUITETURA DE SOFTWARE**  
**DISCIPLINA: Qualidade em  
Arquitetura de Software**  
**Professor: Clávison M. Zapelini**



# Porque é importante usar e conhecer padrões?

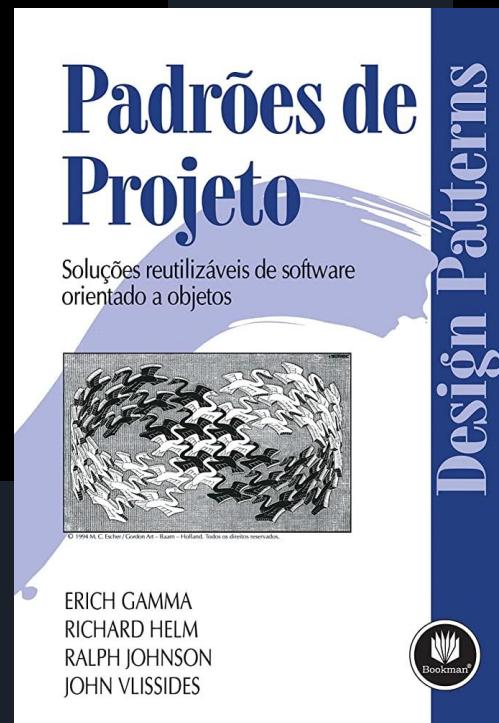
---

- Esse framework é fortemente baseado no padrão Observer....
- Esse framework utiliza o padrão adapter....



Já sabe uma série de  
“COISAS” importantes

GOF  
Gang of Four



- Como se comunica.
- Qual problema o padrão visa resolver.
- Implementação de referência do padrão.

## PADRÕES DE PROJETOS CRIACIONAIS

Mecanismos para criar um objeto. Ao invés de utilizar diretamente o operador "new" , pode-se utilizar algum padrão que forneça mais flexibilidade no código. [@Autowied](#)

## PADRÕES DE PROJETOS ESTRUTURAIS

Mostrar como objetos podem se unir em estruturas maiores, porém de forma organizada, facilitando possíveis extensões. [Adapter](#)

## PADRÕES DE PROJETOS COMPORTAMENTAIS

Organizar a forma de comunicação entre os objetos. [Acoplamento, dependências](#)



Cada padrão descreve um problema que ocorre frequentemente em seu ambiente, e então descreve o cerne da solução para aquele problema, de um modo tal que você pode usar esta solução milhões de vezes, sem nunca fazer a mesma coisa repetida

**CHRISTOPHER ALEXANDER**





# MÓDULO PADRÕES DE PROJETOS CRIACIONAIS

MUITAS VEZES CRIAR OBJETOS PODE SER  
COMPLICADO E POR ESSE MOTIVO EXISTEM  
PADRÕES PRA AJUDAR!



# FACTORY METHOD

CHEGOU A HORA DE CONHECER O NOSSO  
PRIMEIRO PADRÃO, E UM DOS MAIS  
FAMOSOS!



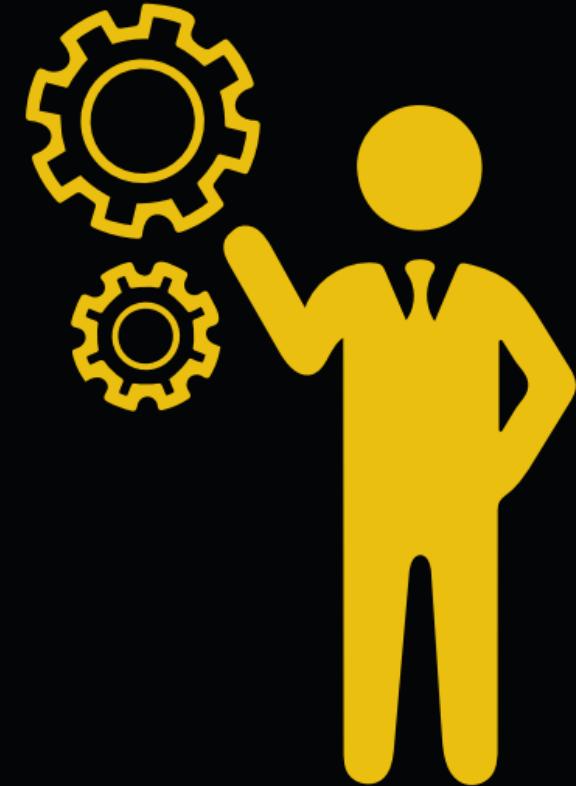
# PROBLEMAS

- COMO POSSO ESCREVER UM CÓDIGO ONDE AS CLASSES INSTANCIADAS POSSAM VARIAR DENTRO DE UMA MESMA INTERFACE?
- COMO DEIXAR O MEU CÓDIGO DESACOPLADO DAS CLASSES CONCRETAS?



# SOLUÇÃO

- EXTRAIR A LÓGICA DE CRIAÇÃO DOS OBJETOS PARA UM FACTORY METHOD
- INVOCAR O FACTORY METHOD PARA RECEBER UMA INSTÂNCIA QUALQUER QUE IMPLEMENTE UMA DETERMINADA INTERFACE





Um padrão que define uma interface para criar um objeto, mas permite às classes decidirem qual classe instanciar.

O Factory Method permite a uma classe deferir a instanciação para subclasses.

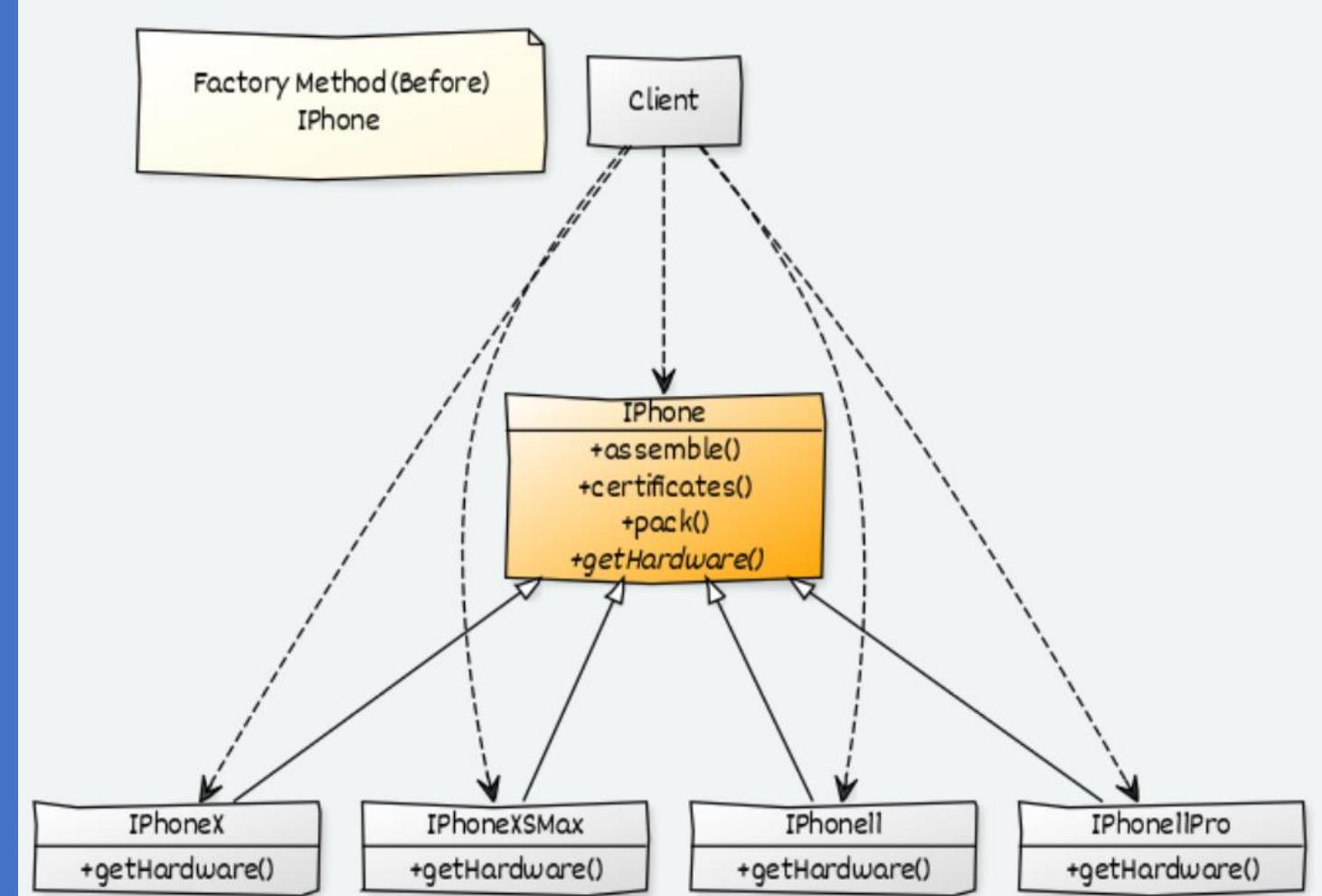


GOF

Exemplo Monstro

```
public IPhone orderIPhone() {  
    IPhone device = null;  
  
    //...  
  
    device = new IPhoneX();  
  
    //...  
  
    device = new IPhone11();  
  
    //...  
  
    return device;  
}
```

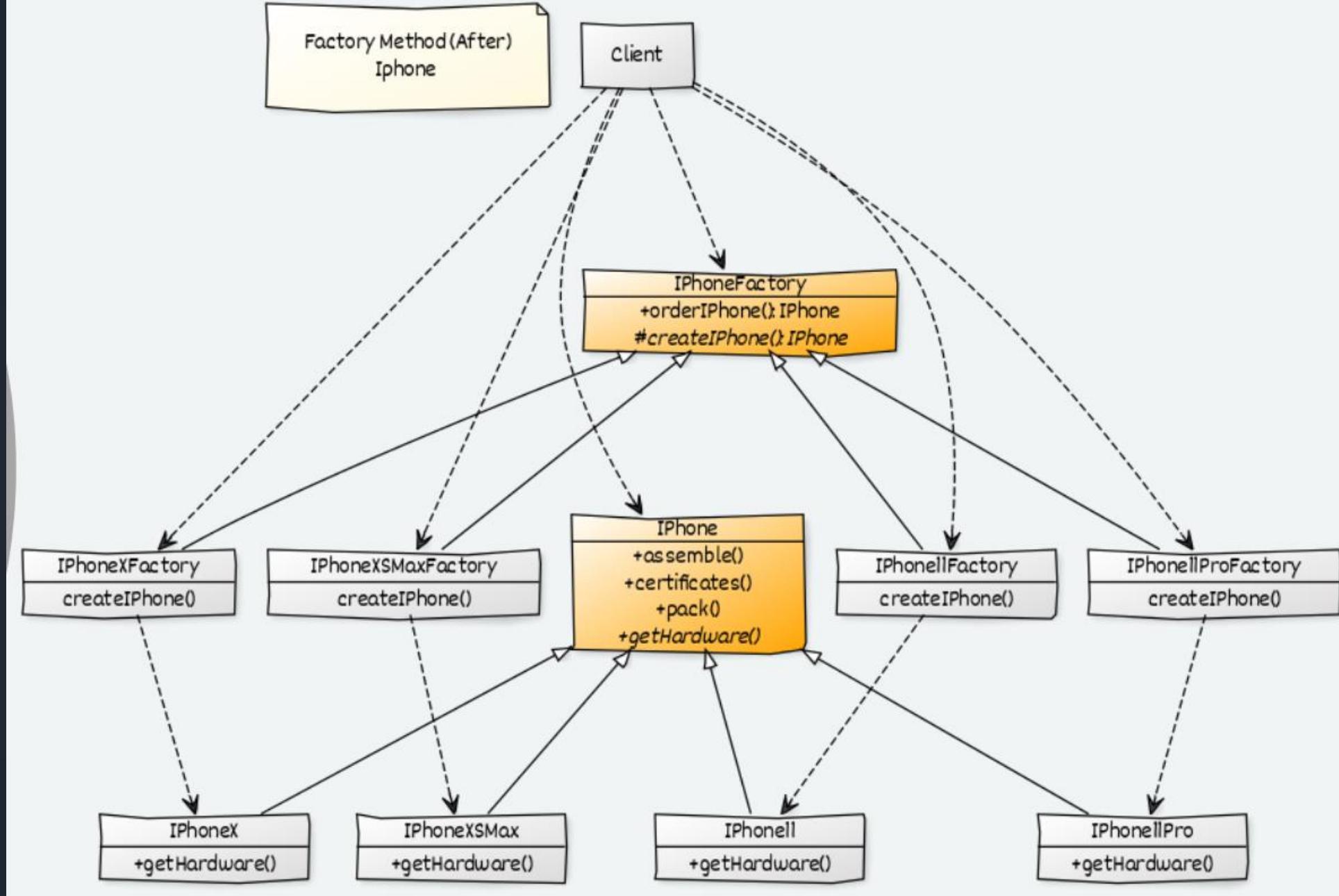
## SEM FACTORY METHOD



**Não é a  
forma  
mais  
utilizada**

Um FACTORY para cada classe concreta  
Muita complexidade

```
public IPhone orderIPhone() {  
    IPhone device = null;  
    IPhoneFactory iphoneXFactory = new IPhoneXFactory();  
    IPhoneFactory iphone11ProFactory = new IPhone11ProFactory();  
  
    //...  
  
    device = iphoneXFactory.orderIPhone();  
  
    //...  
  
    device = iphone11ProFactory.orderIPhone();  
    return device;  
}
```

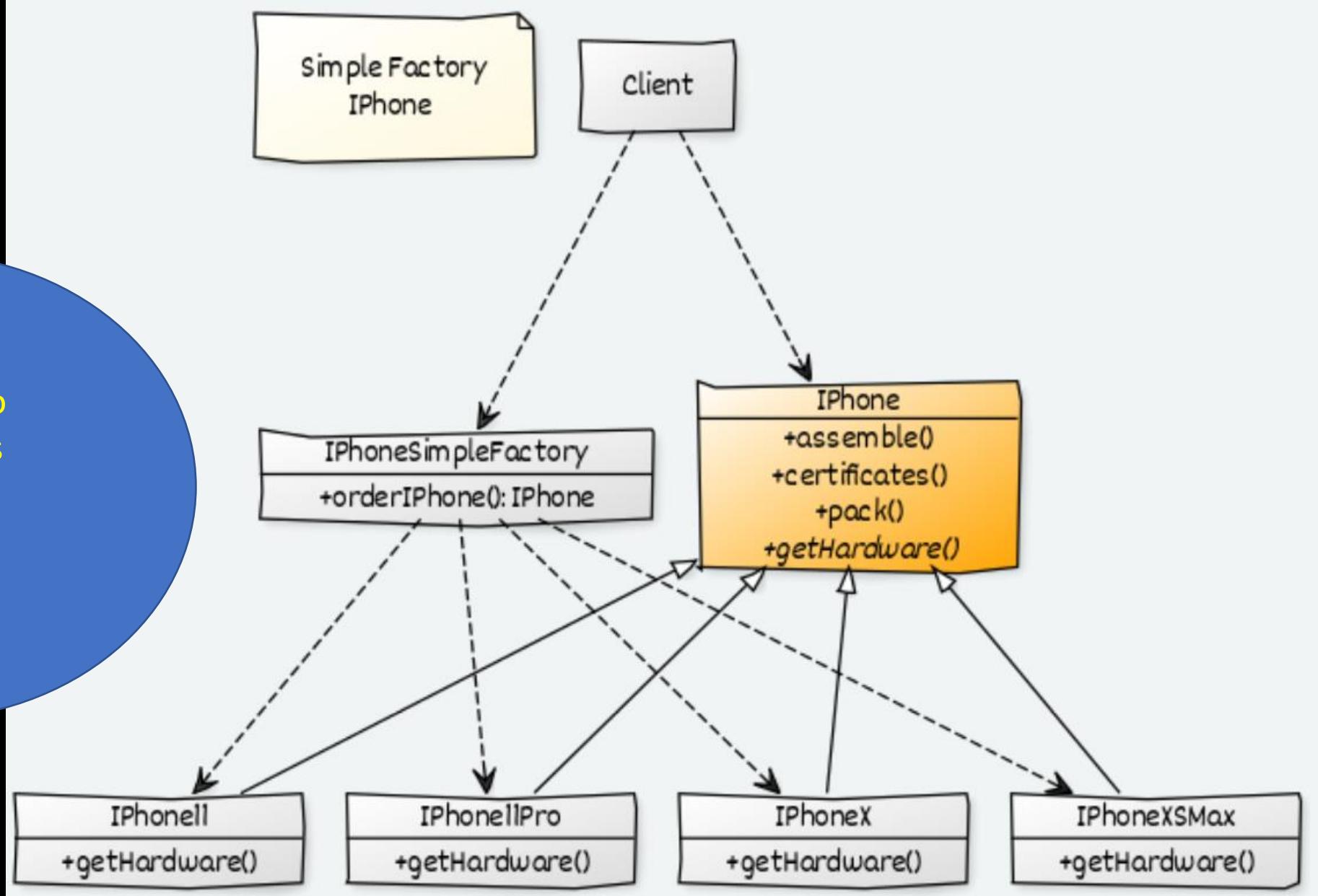


# SIMPLE FACTORY

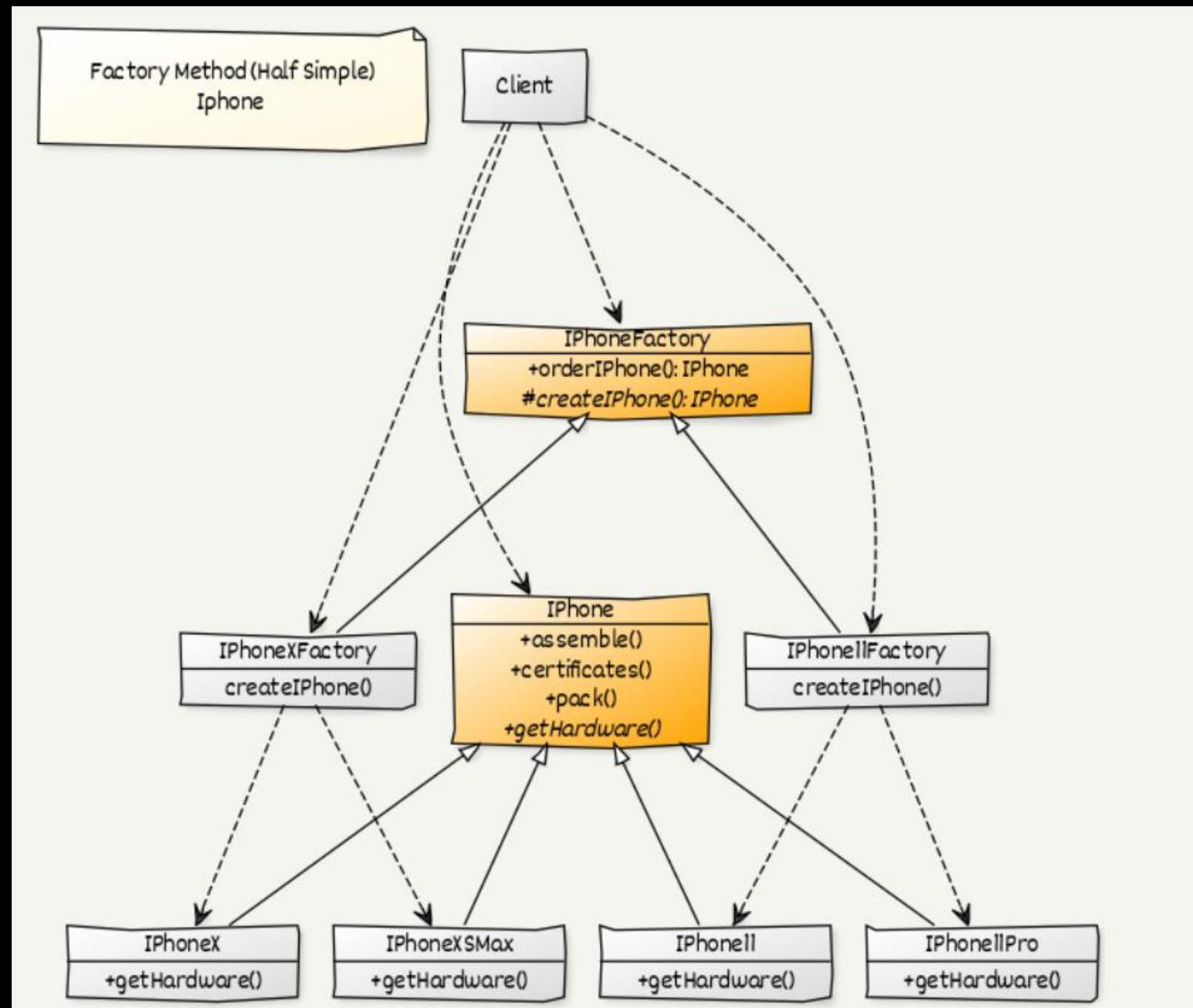
## Um FACTORY genérico

```
public IPhone orderIPhone() {  
    IPhone device = null;  
  
    //...  
  
    device = IPhoneSimpleFactory.orderIPhone("X", "standard");  
  
    //...  
  
    device = IPhoneSimpleFactory.orderIPhone("11", "highEnd");  
  
    //...  
  
    return device;  
}
```

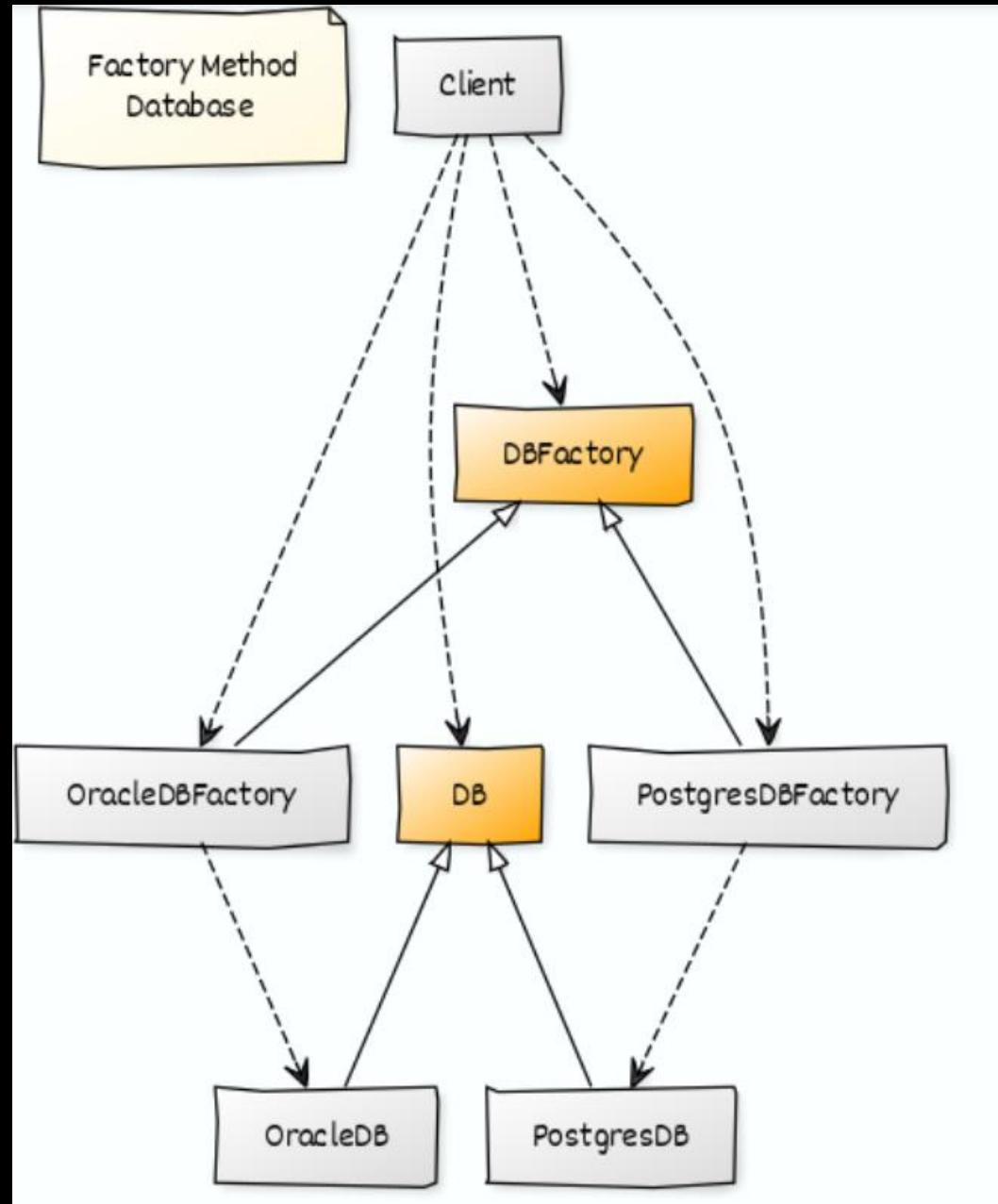
Cliente não  
conhece as  
classes  
concretas.



## Outra arquitetura....



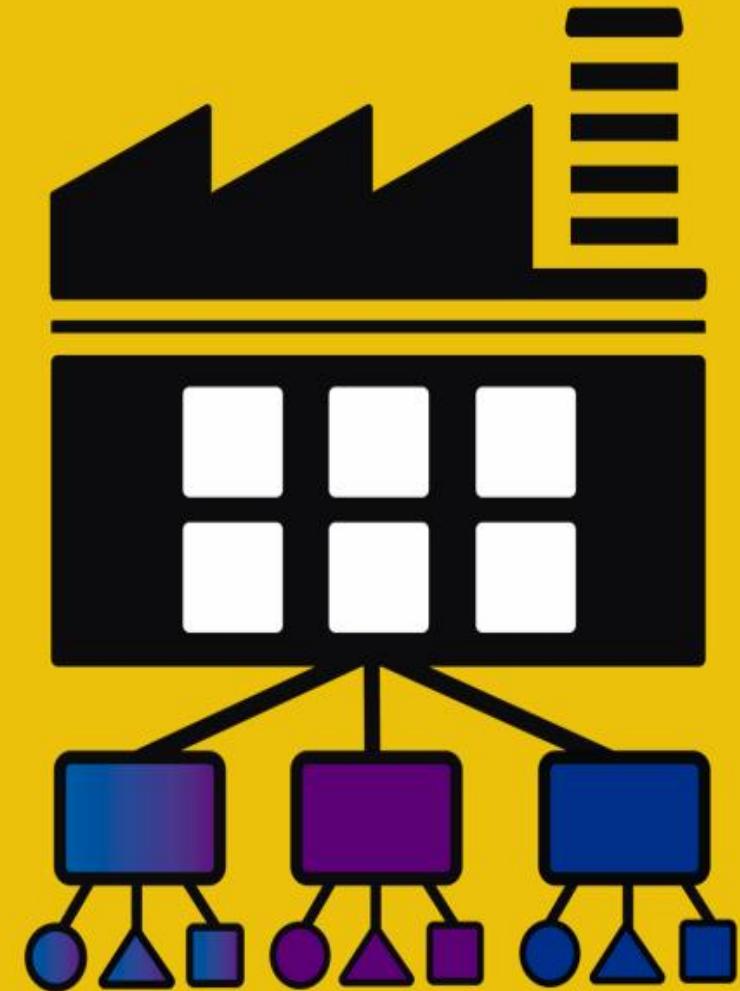
Mais outra ...





# ABSTRACT FACTORY

ESSE SIM É UM CARA DE FAMÍLIA



# PROBLEMAS

- COMO POSSO ESCREVER UM CÓDIGO ONDE AS CLASSES INSTANCIADAS POSSAM VARIAR DENTRO DE UMA MESMA INTERFACE?
- COMO GARANTIR QUE UM CONJUNTO DE OBJETOS RELACIONADOS (OU DEPENDENTES) POSSAM SER CRIADOS MANTENDO O CONTEXTO ÚNICO?



# SOLUÇÃO

- EXTRAIR A LÓGICA DE CRIAÇÃO DOS OBJETOS PARA UM ABSTRACT FACTORY
- CRIAR UMA IMPLEMENTAÇÃO DO ABSTRACT FACTORY PARA CADA CONTEXTO, GARANTINDO QUE TODOS OS OBJETOS CRIADOS ESTEJAM RELACIONADOS

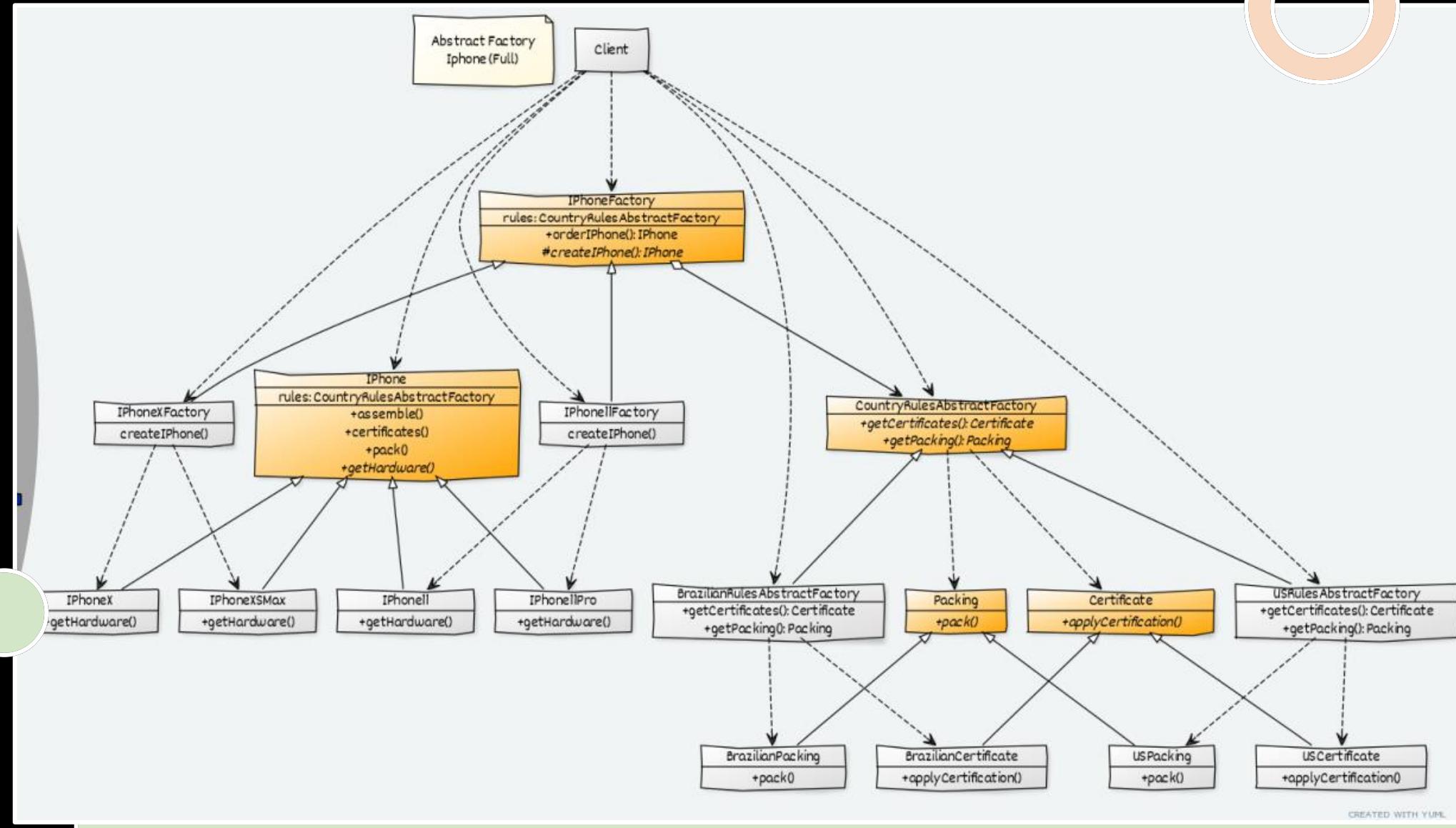




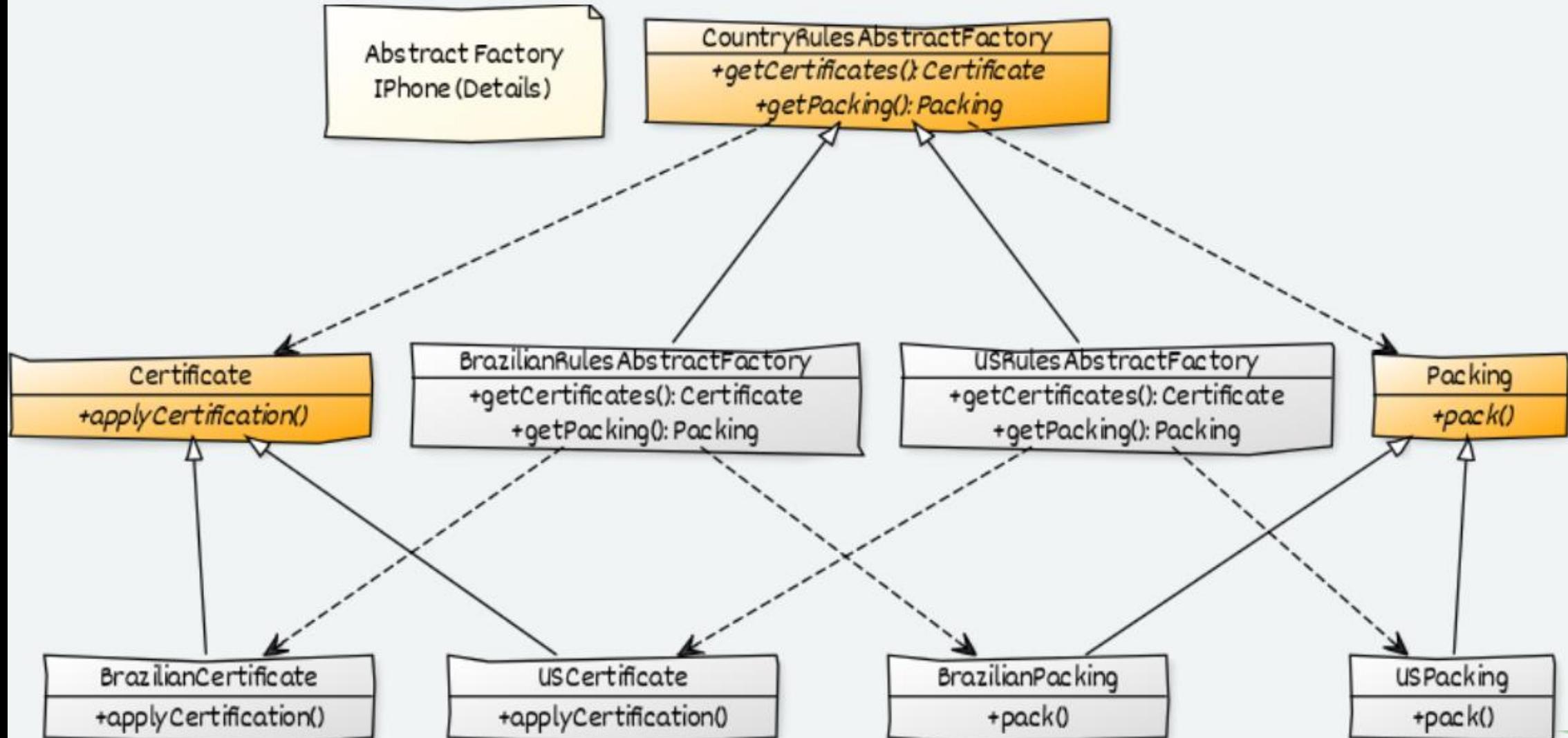
Prover uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

GOF



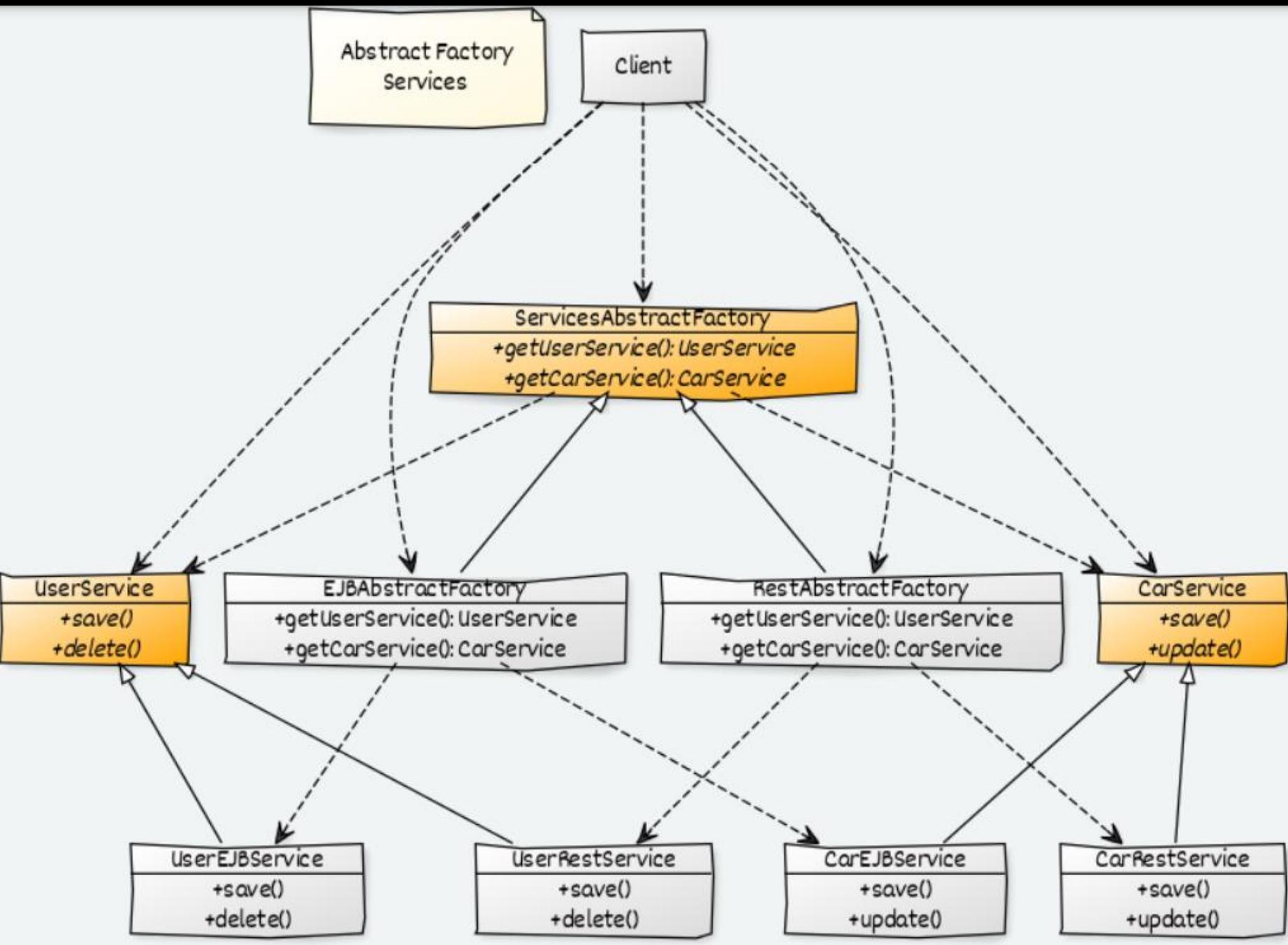


CREATED WITH YUML

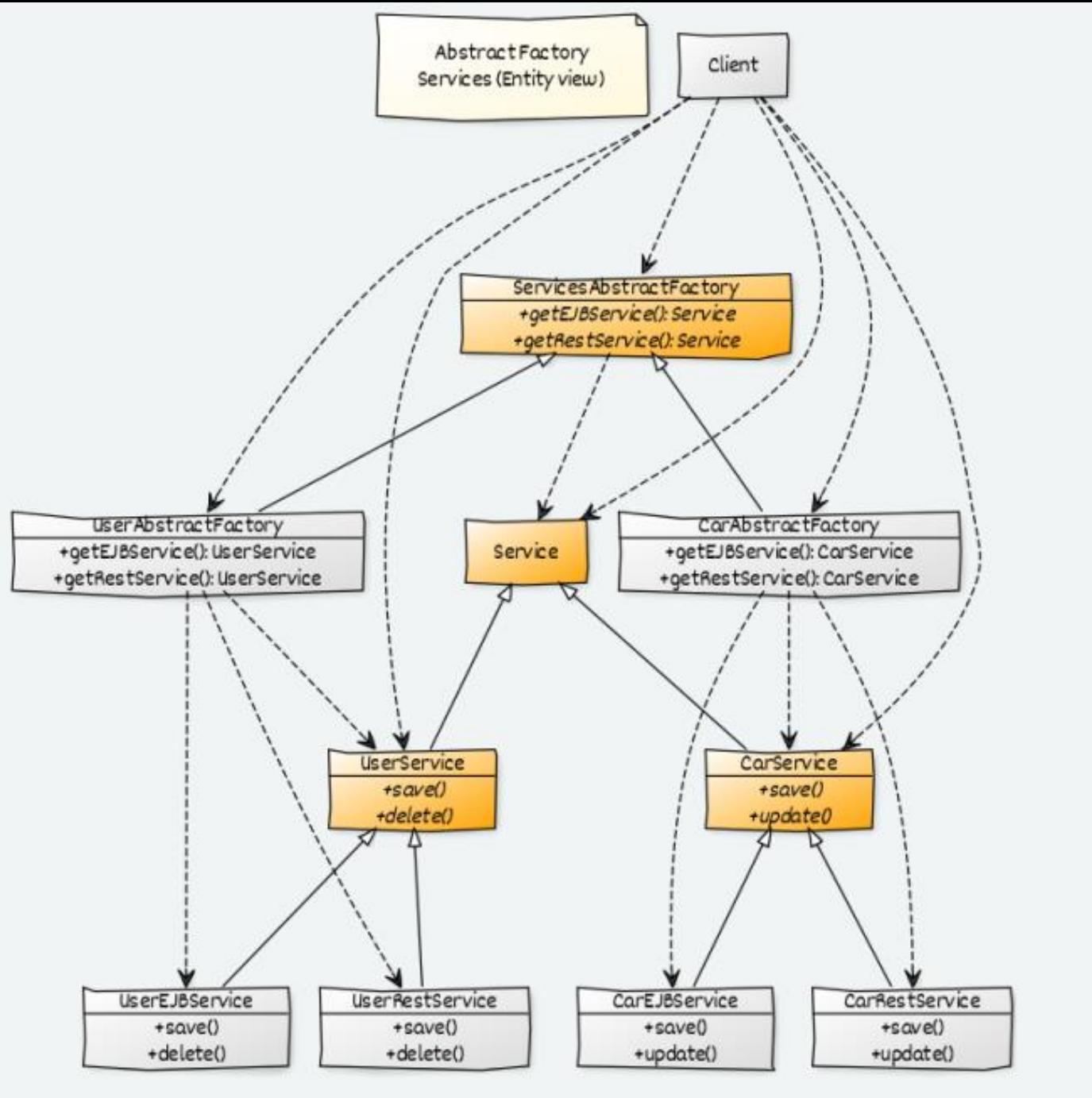


```
public abstract class IPhone {  
    CountryRulesAbstractFactory rules;  
  
    public IPhone(CountryRulesAbstractFactory rules) {  
        this.rules = rules;  
    }  
  
    //...  
  
    public void certificates() {  
        System.out.println("Testing all the certificates");  
        System.out.println(rules.getCertificates().applyCertification());  
    }  
  
    public void pack() {  
        System.out.println("Packing the device");  
        System.out.println(rules.getPacking().pack());  
    }  
}
```

## Outra arquitetura....



Mais outra ...



# SINGLETON

UM PADRÃO SIMPLES E POLÊMICO.



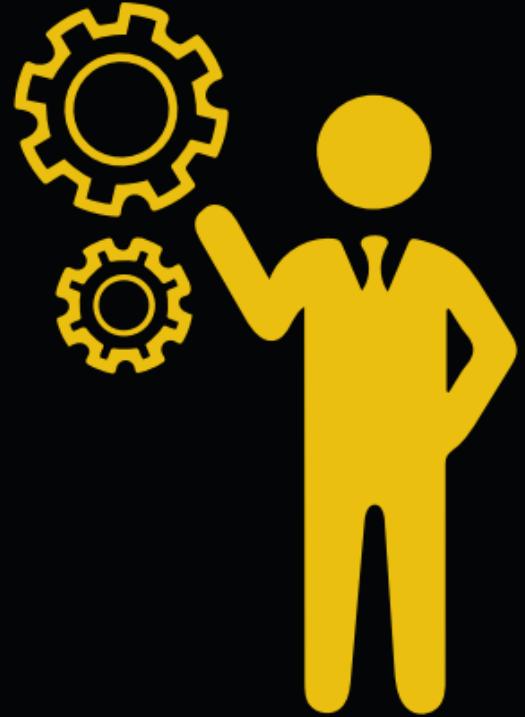
# PROBLEMAS

- COMO POSSO GARANTIR QUE UMA CLASSE TENHA APENAS UMA INSTÂNCIA?
- COMO FAZER COM QUE ESTA INSTÂNCIA ÚNICA POSSA SER ACESSÍVEL GLOBALMENTE?



# SOLUÇÃO

- ESCONDER O CONSTRUTOR DESSA CLASSE...
- DEFINIR UM PONTO DE CRIAÇÃO ESTÁTICO...
- ... QUE RETORNE ESTA INSTÂNCIA ÚNICA

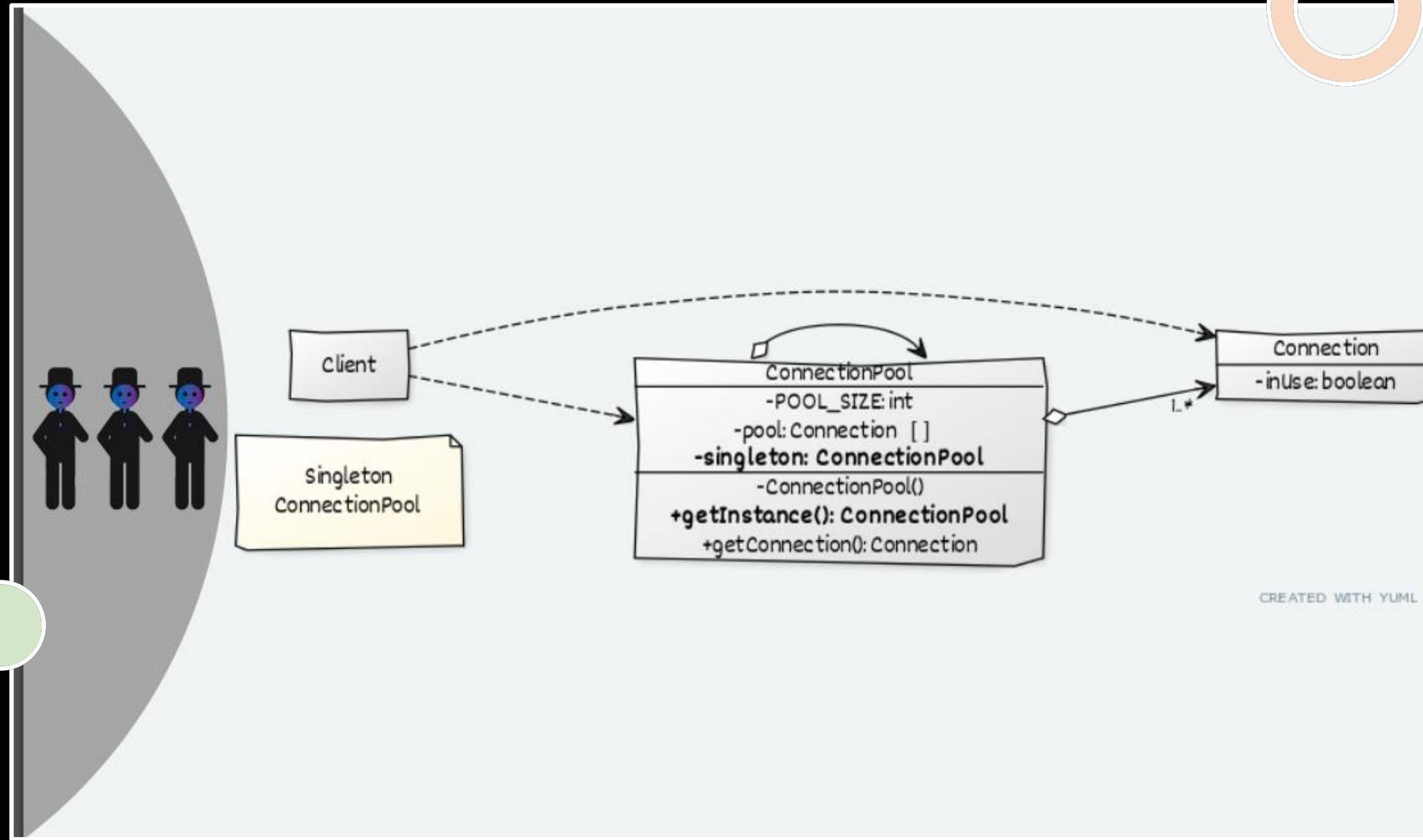




Garantir que uma classe só tenha uma única instância, e prover um ponto de acesso global a ela.

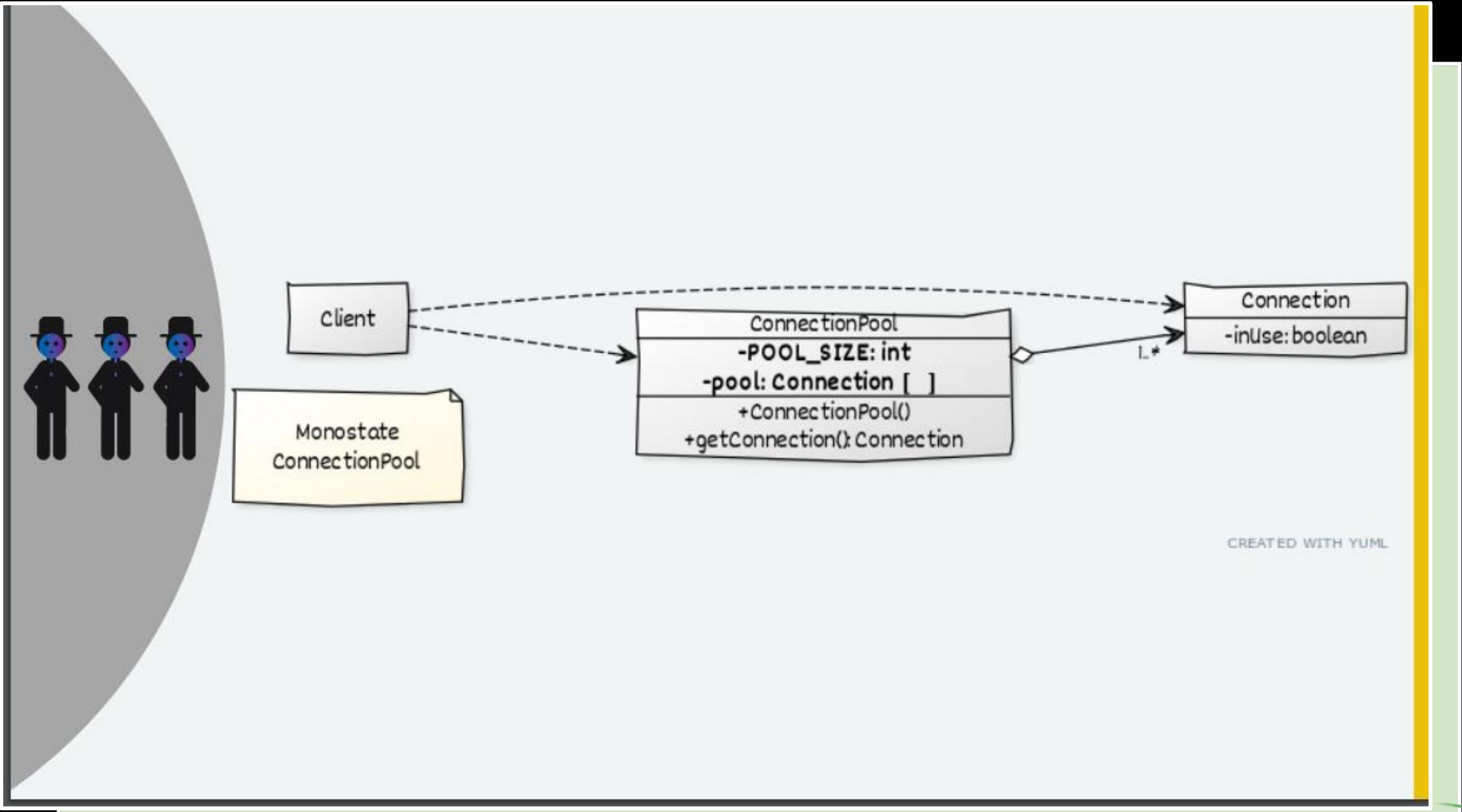
**GOF**





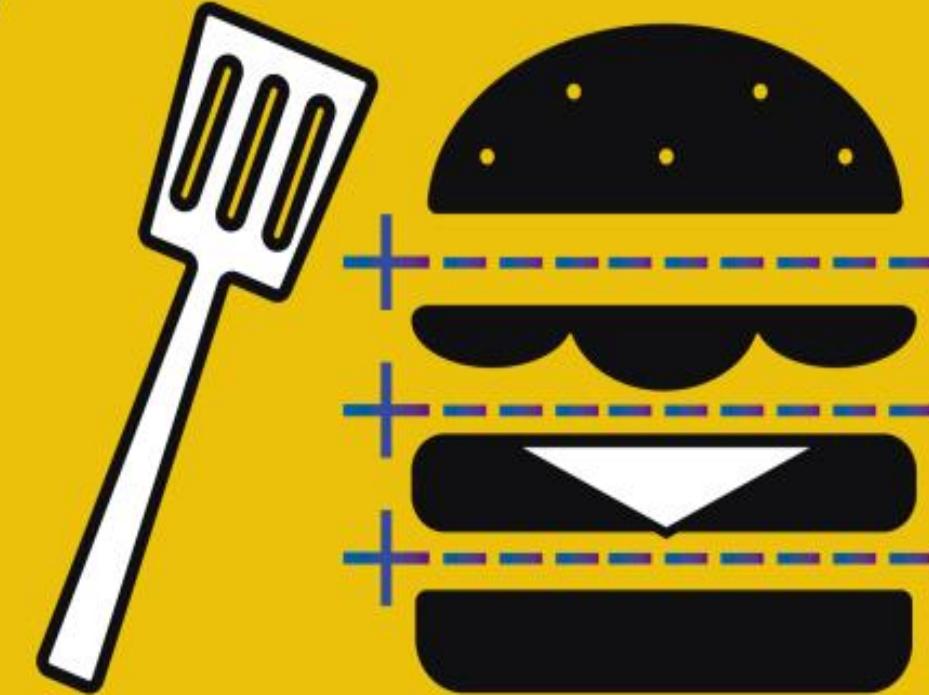
CREATED WITH YUML

```
public class ConnectionPool {  
    private static ConnectionPool singleton = new ConnectionPool();  
    private List<Connection> connectionsPool;  
  
    public static ConnectionPool getInstance() {  
        return singleton;  
    }  
  
    private ConnectionPool() {  
        System.out.println("Creating Connection Pool");  
        connectionsPool = new ArrayList<Connection>();  
        for(int i = 0; i < POOL_SIZE; i++) {  
            connectionsPool.add(new Connection());  
        }  
    }  
  
    //...  
}
```



# BUILDER

UM PADRÃO CLÁSSICO QUE FOI EVOLUINDO  
COM OUTRAS ABORDAGENS



# PROBLEMAS

- COMO UMA CLASSE PODE CRIAR  
DIFERENTES REPRESENTAÇÕES DE UM  
MESMO OBJETO COMPLEXO?



# SOLUÇÃO

- DELEGAR A CRIAÇÃO DO OBJETO PARA UM BUILDER AO INVÉS DE INSTANCIAR O OBJETO CONCRETO DIRETAMENTE
- DIVIDIR A CRIAÇÃO DO OBJETO EM PARTES...
- ENCAPSULAR A CRIAÇÃO E MONTAGEM DESSAS PARTES EM UM BUILDER SEPARADO

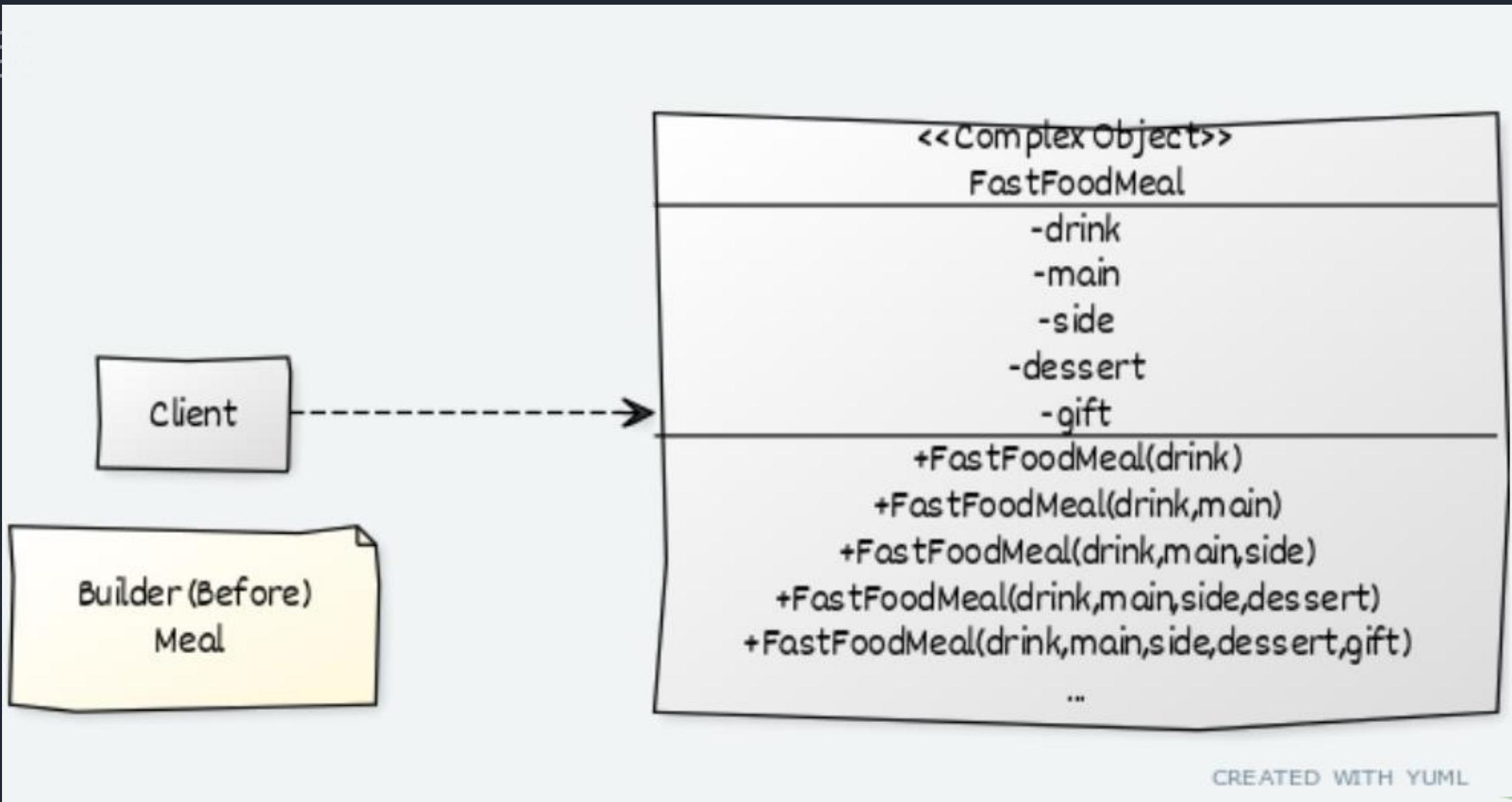




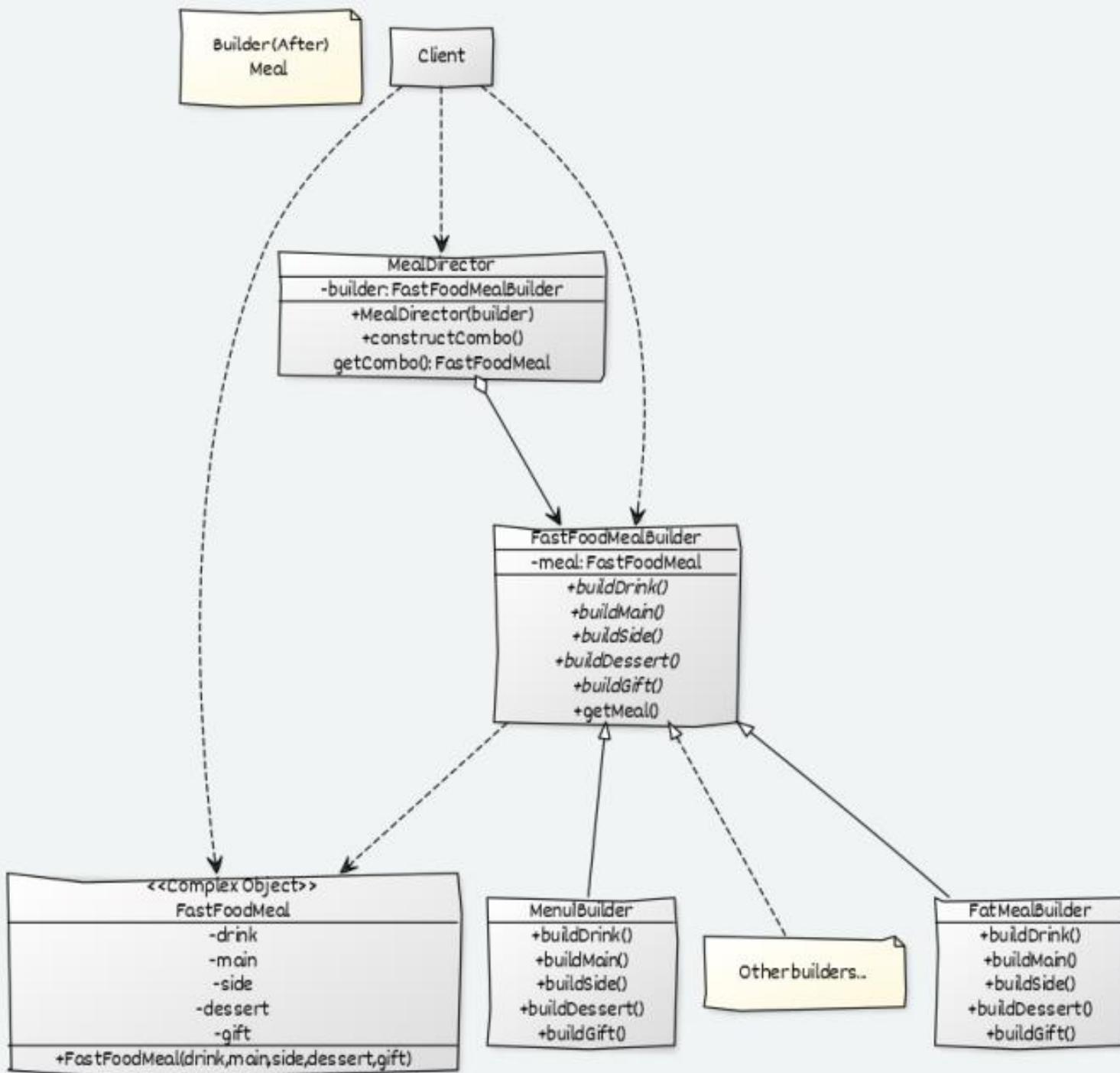
Separar a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar representações diferentes.

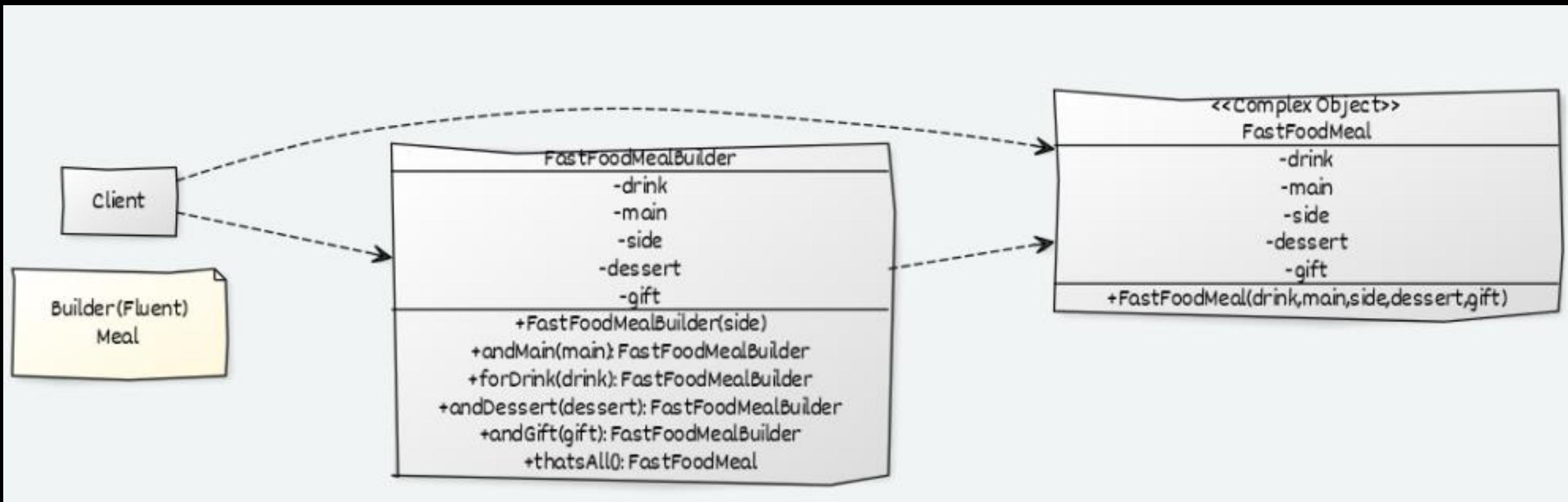
**GOF**

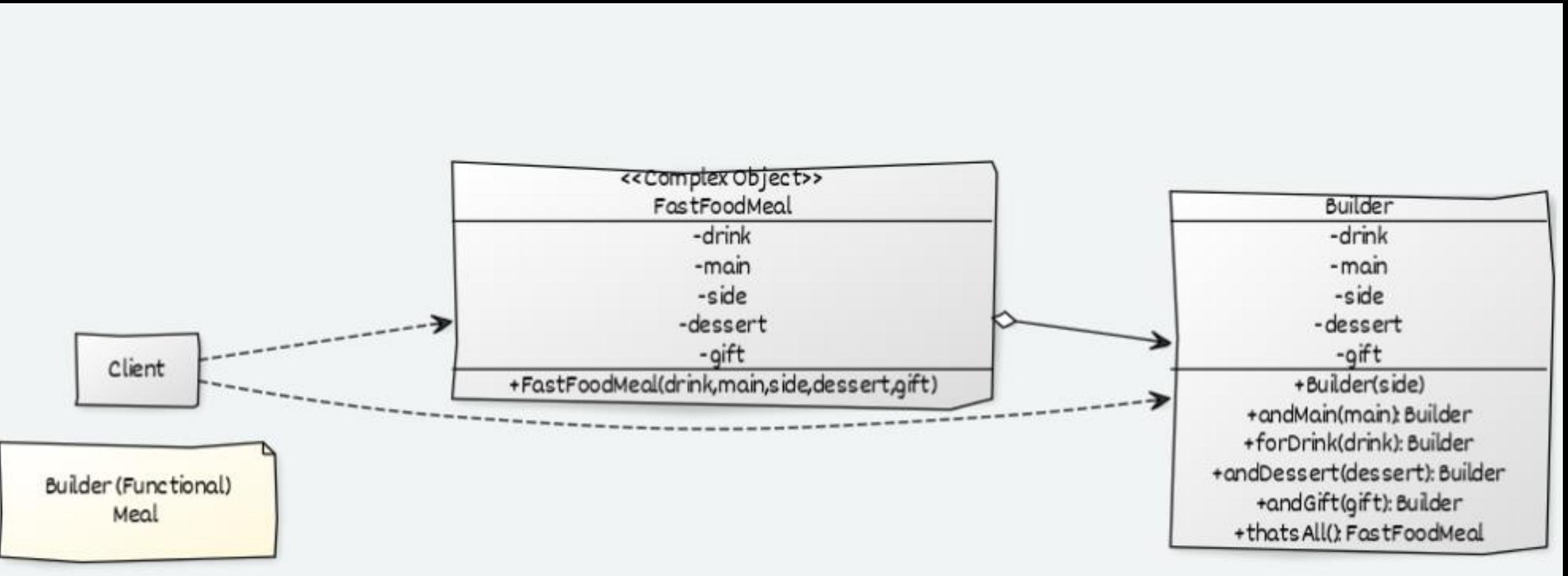




CREATED WITH YUML



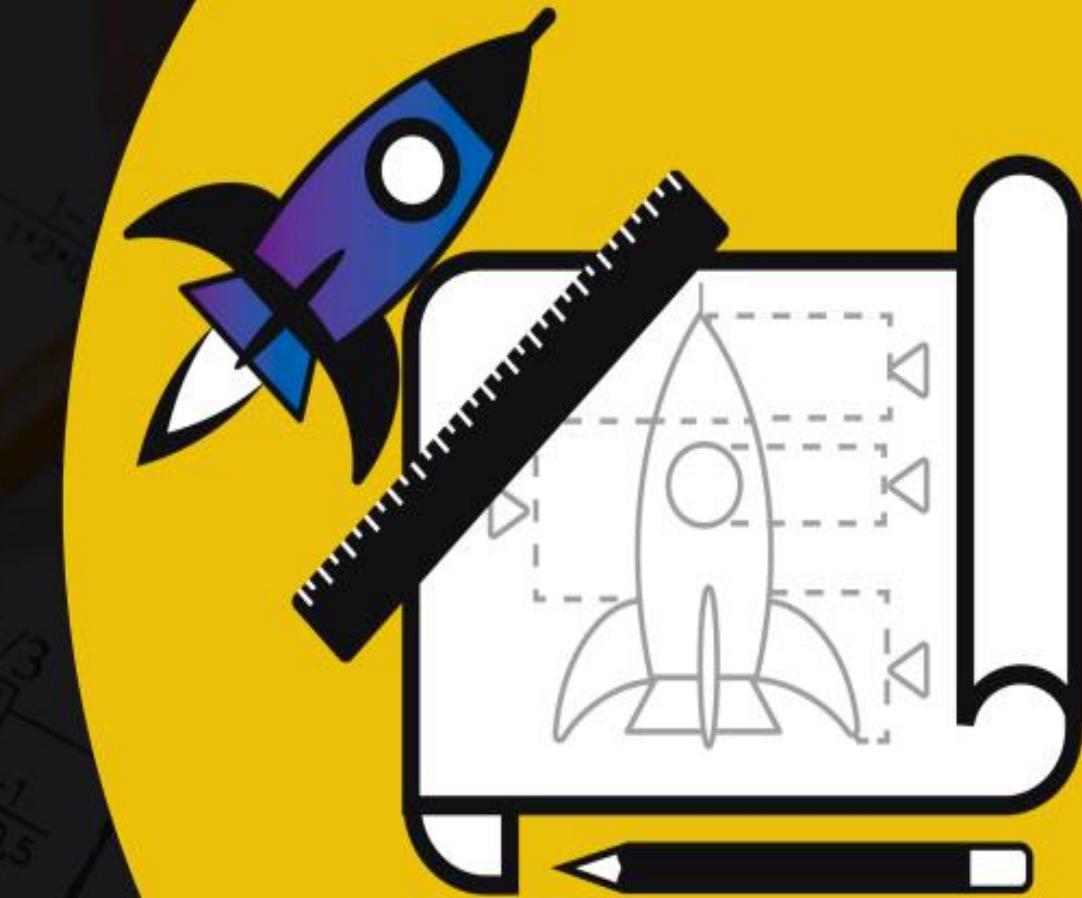




```
public class Client {  
  
    public static void main(String[] args) {  
        //Before  
        justFries = new FastFoodMeal(null, null, "Fries");  
  
        //After  
        justFries = order("Just Fries", new JustFriesBuilder());  
  
        //Fluent  
        justFries = new FastFoodMealBuilder("Fries").thatsAll();  
  
        //Functional  
        justFries = new FastFoodMeal.Builder("Fries").thatsAll();  
    }  
}
```

# PROTOTYPE

RESUMINDO... CLONE!



# PROBLEMAS

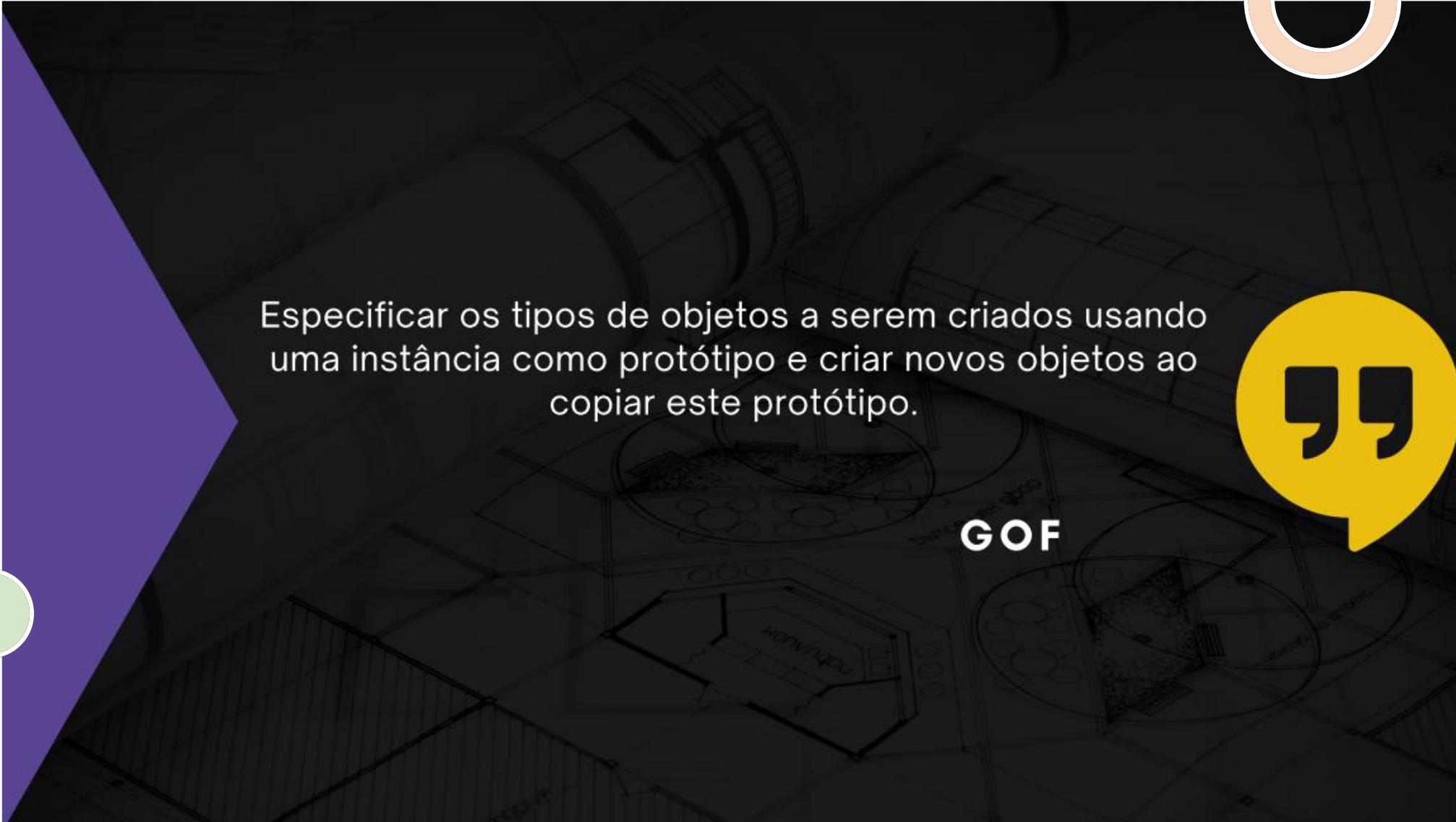
- COMO POSSO CRIAR UM OBJETO NOVO  
APROVEITANDO O ESTADO PREVIAMENTE  
EXISTENTE DE OUTRO OBJETO?



# SOLUÇÃO

- DEFINIR UM PROTOTYPE QUE RETORNE A CÓPIA DE SI MESMO

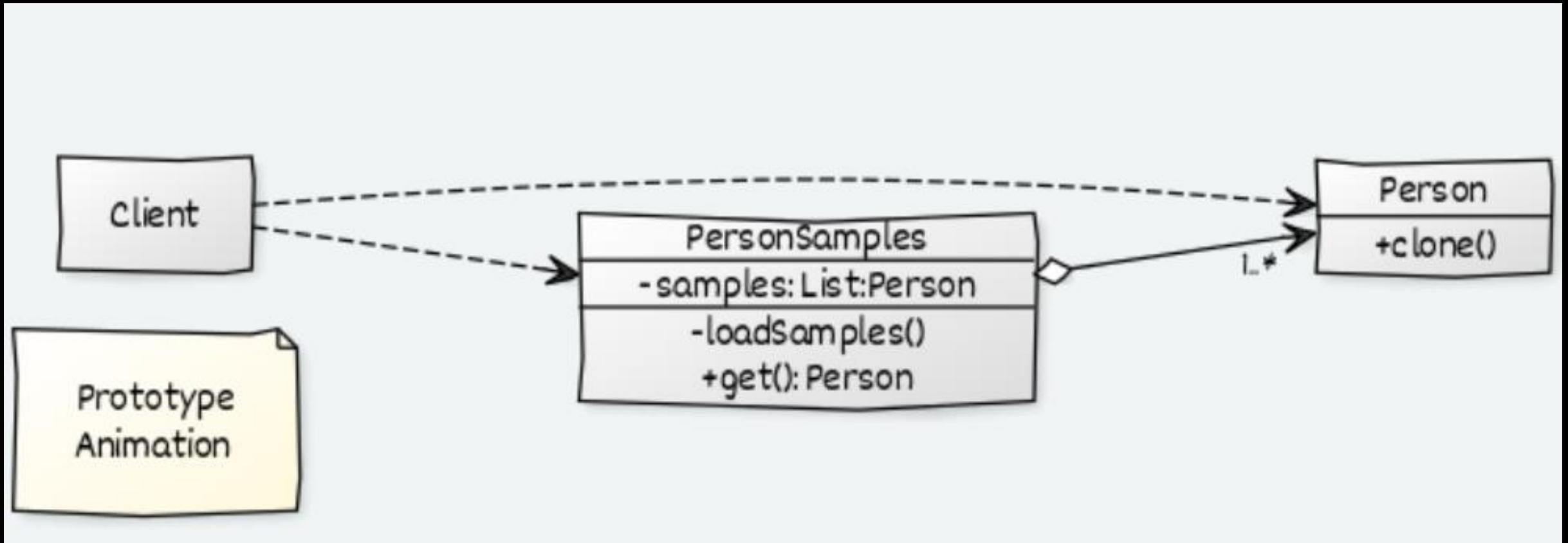


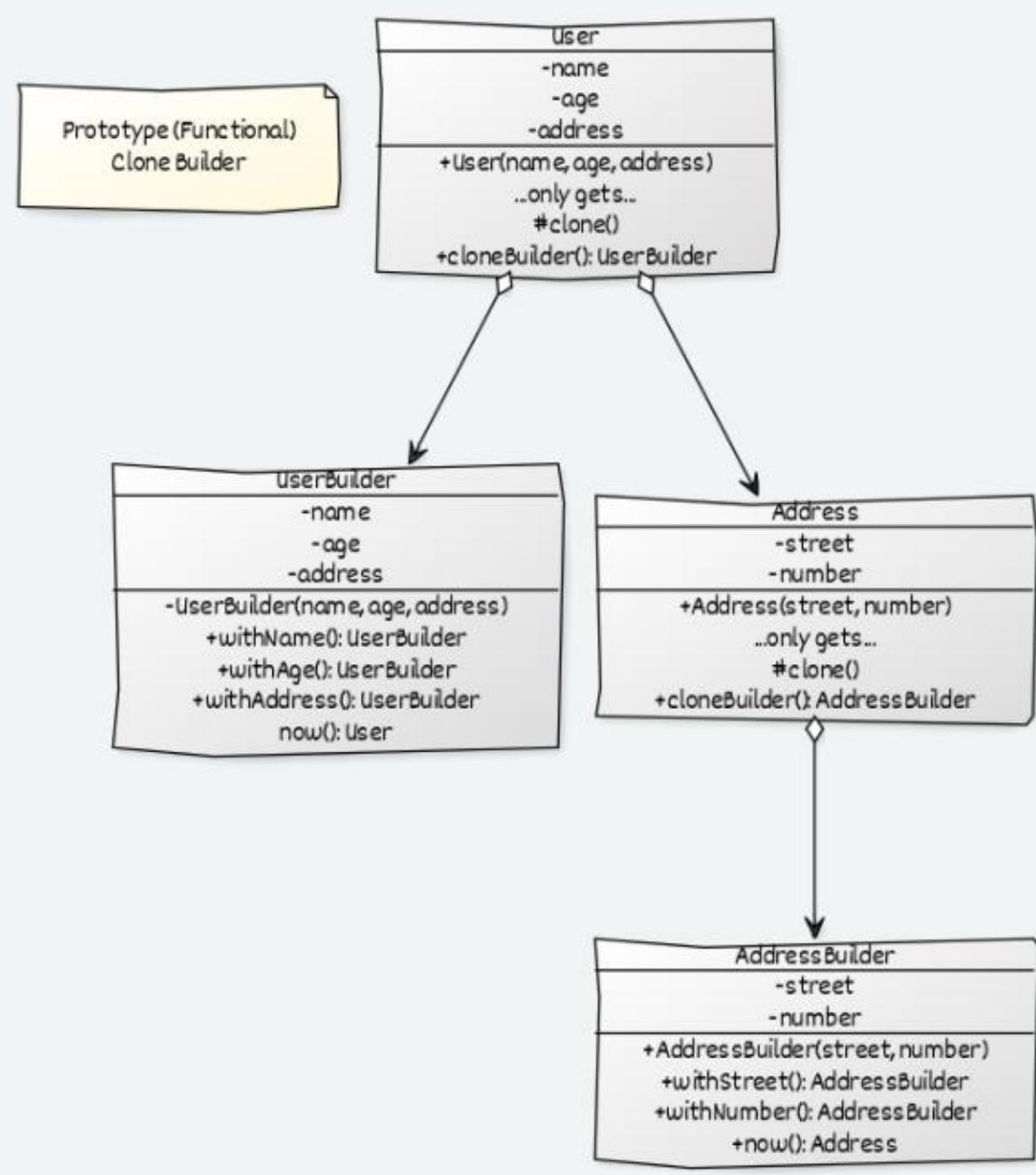


Especificar os tipos de objetos a serem criados usando uma instância como protótipo e criar novos objetos ao copiar este protótipo.

**GOF**





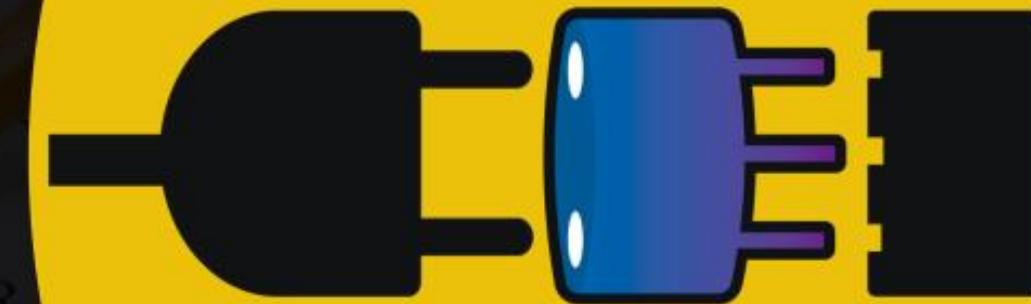


# MÓDULO ← PADRÕES DE PROJETOS ESTRUTURAIS

COMO SERIA POSSÍVEL ESTRUTURAR DIVERSOS  
OBJETOS E CLASSES DE FORMA EXTENSÍVEL E  
FLEXÍVEL?

# ADAPTER

ADAPTANDO BEM, TODO MUNDO SE INTEGRA



# PROBLEMAS

- COMO UMA CLASSE PODE SER REUSADA MESMO QUE NÃO TENHA A INTERFACE REQUISITADA PELO CLIENTE?
- COMO CLASSES DE INTERFACES INCOMPATÍVEIS PODEM TRABALHAR JUNTAS?



# SOLUÇÃO

- DEFINIR UMA CLASSE ADAPTER QUE CONVERTA A INTERFACE DE UMA CLASSE EM OUTRA QUE O CLIENTE NECESSITE

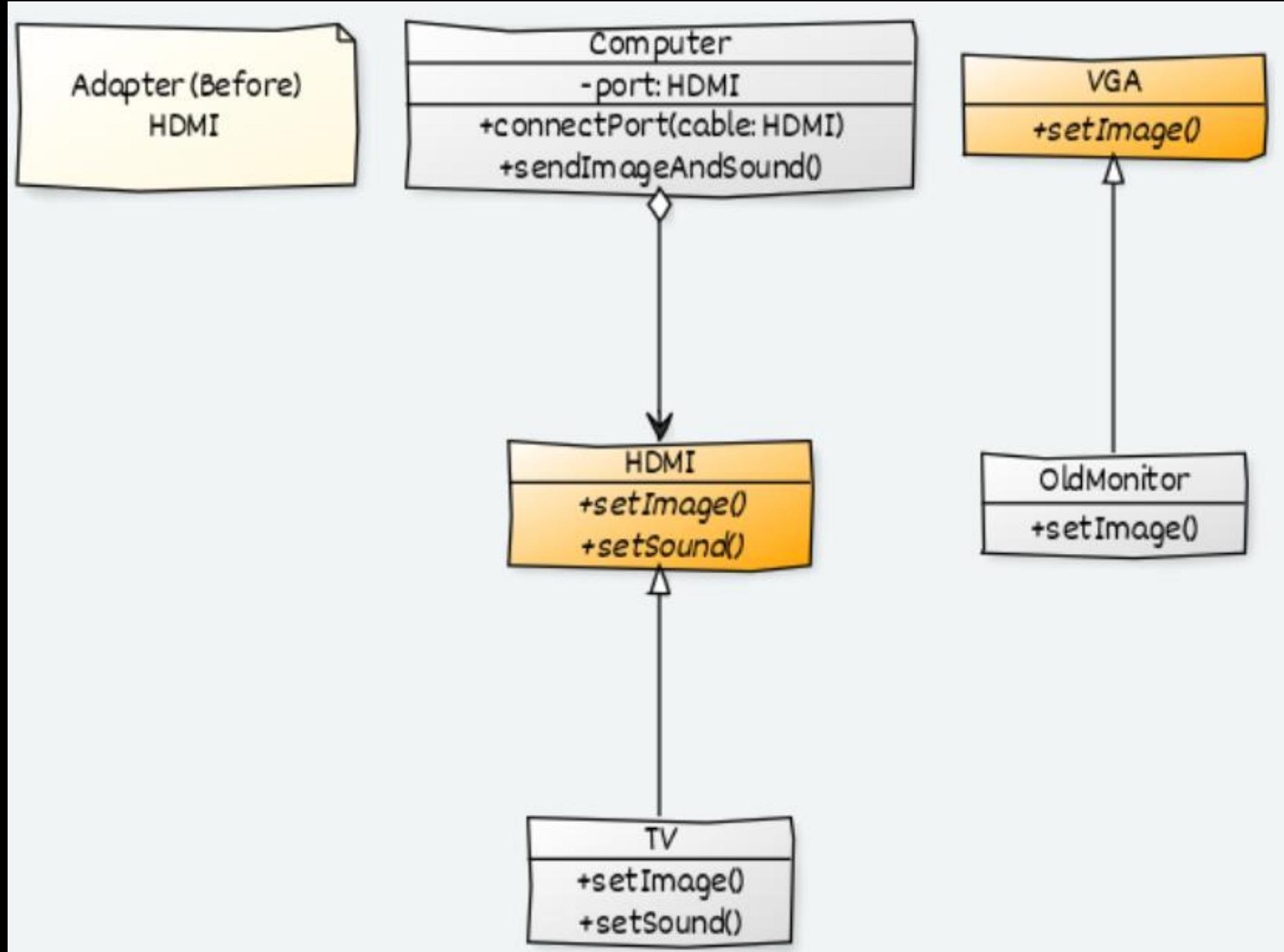


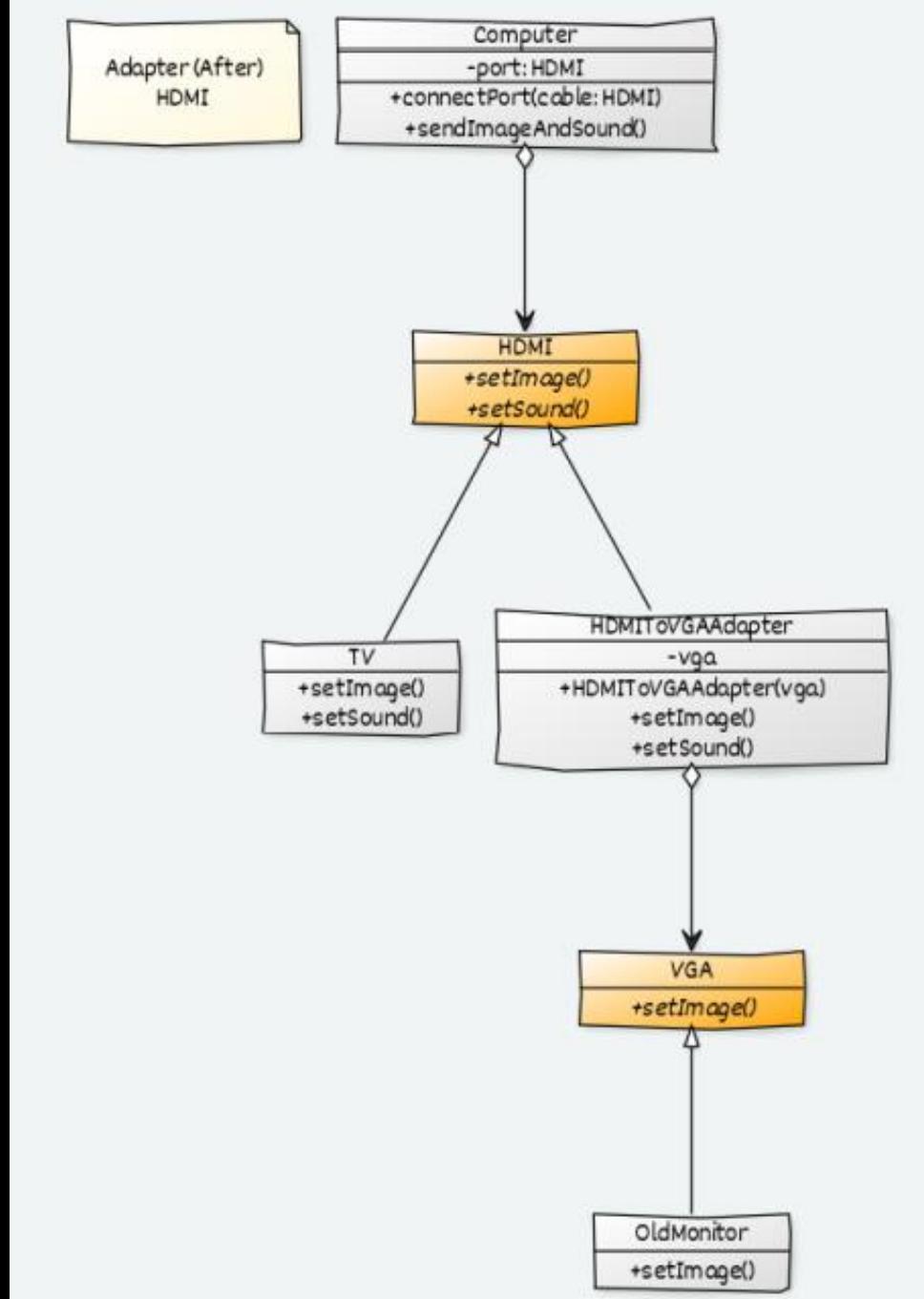


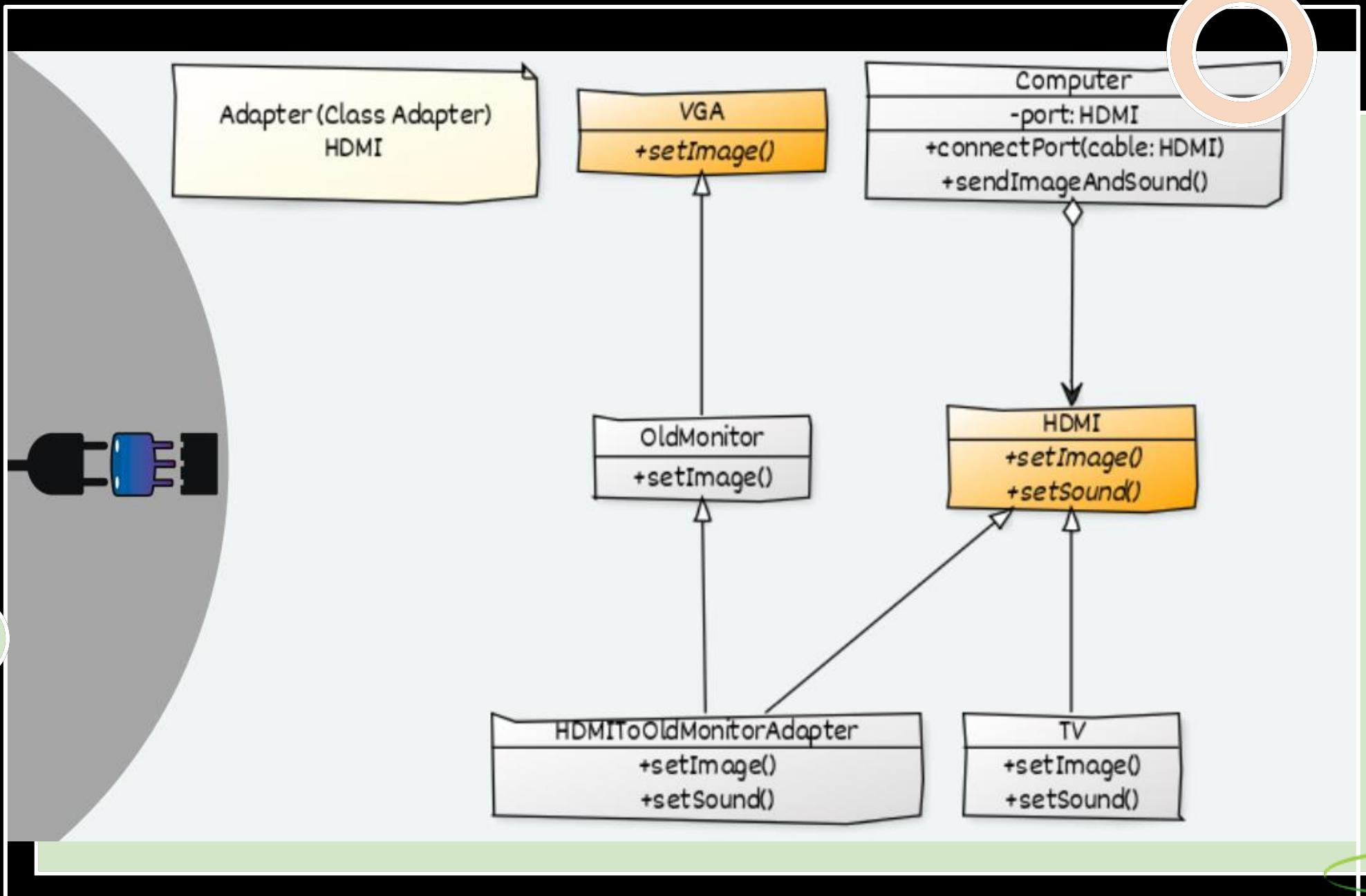
Converter a interface de uma classe em outra interface esperada pelos clientes. Adapter permite a comunicação entre classes que não poderiam trabalhar juntas devido à incompatibilidade de suas interfaces.



**GOF**









# BRIDGE

PONTES SÃO IMPORTANTES.  
NÃO QUEIME PONTES!



# PROBLEMAS

- COMO É POSSÍVEL FAZER COM QUE A ABSTRAÇÃO E A IMPLEMENTAÇÃO POSSAM VARIAR INDEPENDENTEMENTE?
- COMO ESTA IMPLEMENTAÇÃO PODE VARIAR EM TEMPO DE EXECUÇÃO?



# SOLUÇÃO

- DEFINIR UM CONJUNTO HIERÁRQUICO PARA AMBOS OS LADOS: ABSTRAÇÃO E IMPLEMENTAÇÃO
- EM TEMPO DE EXECUÇÃO, SERÁ POSSÍVEL ESCOLHER A CLASSE CONCRETA PARA A ABSTRAÇÃO E PARA A IMPLEMENTAÇÃO...
- ... QUE SÃO COMPATÍVEIS GRAÇAS ÀS INTERFACES

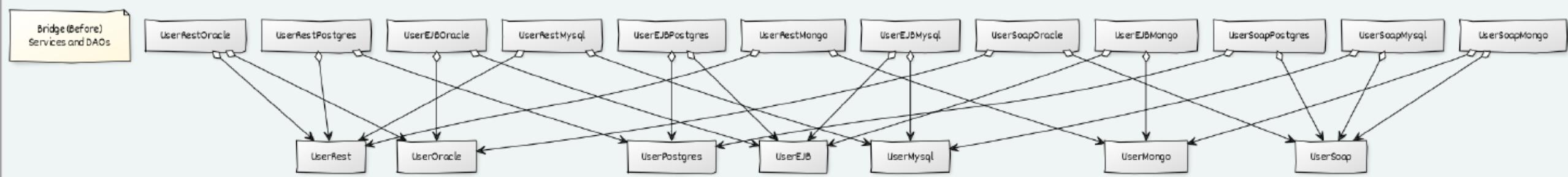
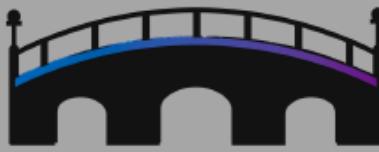




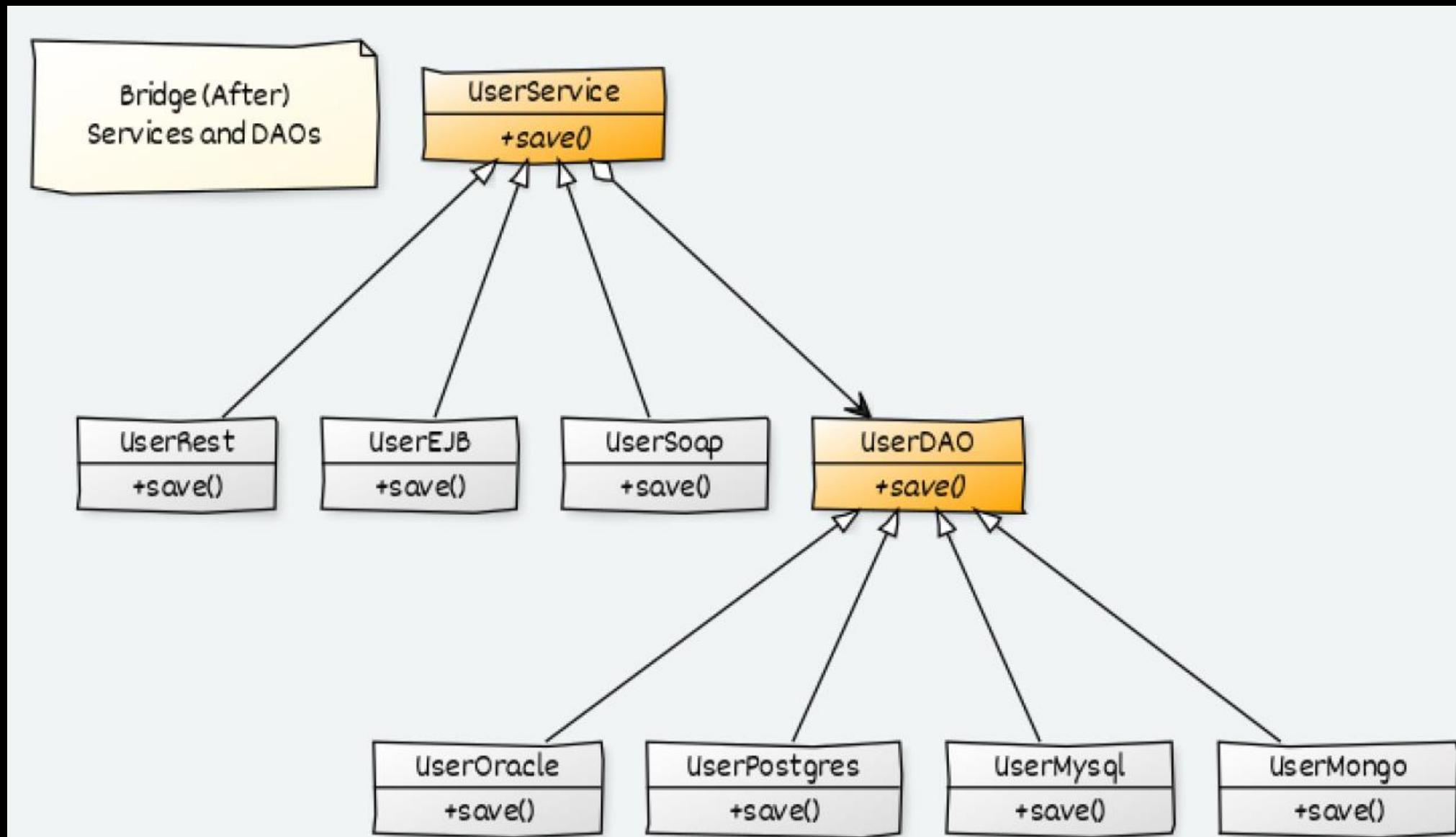
Desacoplar uma abstração de sua implementação para que os dois possam variar independentemente.

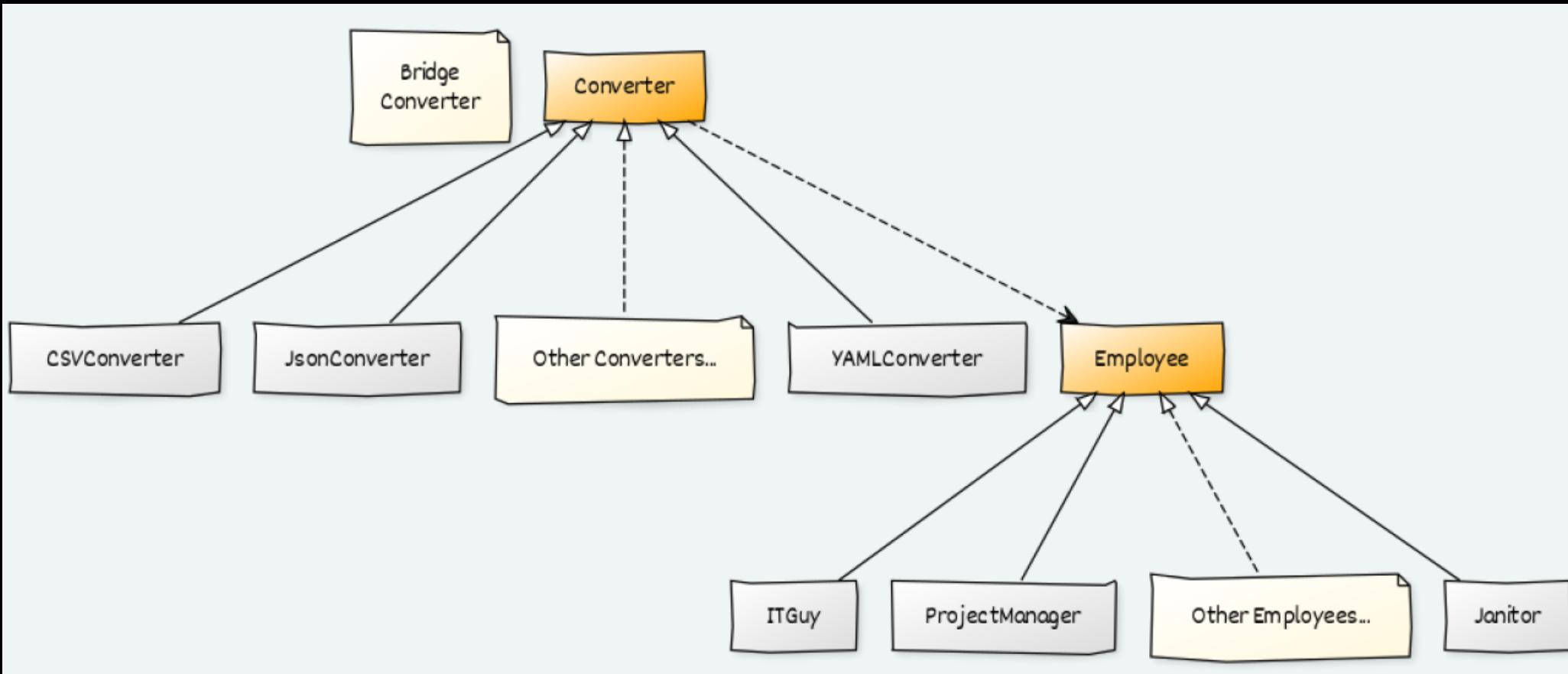
GOF





CREATED WITH YUML





# COMPOSITE

RECURSÃO... RECURSÃO EVERYWHERE



# PROBLEMAS

- COMO SERIA POSSÍVEL CRIAR UMA ESTRUTURA ONDE UMA UNIDADE E UM CONJUNTO DELAS POSSAM SER TRATADAS DE FORMA TRANSPARENTE, SEM DIFERENÇAS?



# SOLUÇÃO

- DEFINIR UMA COMPOSIÇÃO QUE POSSA COMPORTAR TANTO UM ELEMENTO APENAS QUANDO UM CONJUNTO DESSES MESMOS ELEMENTOS

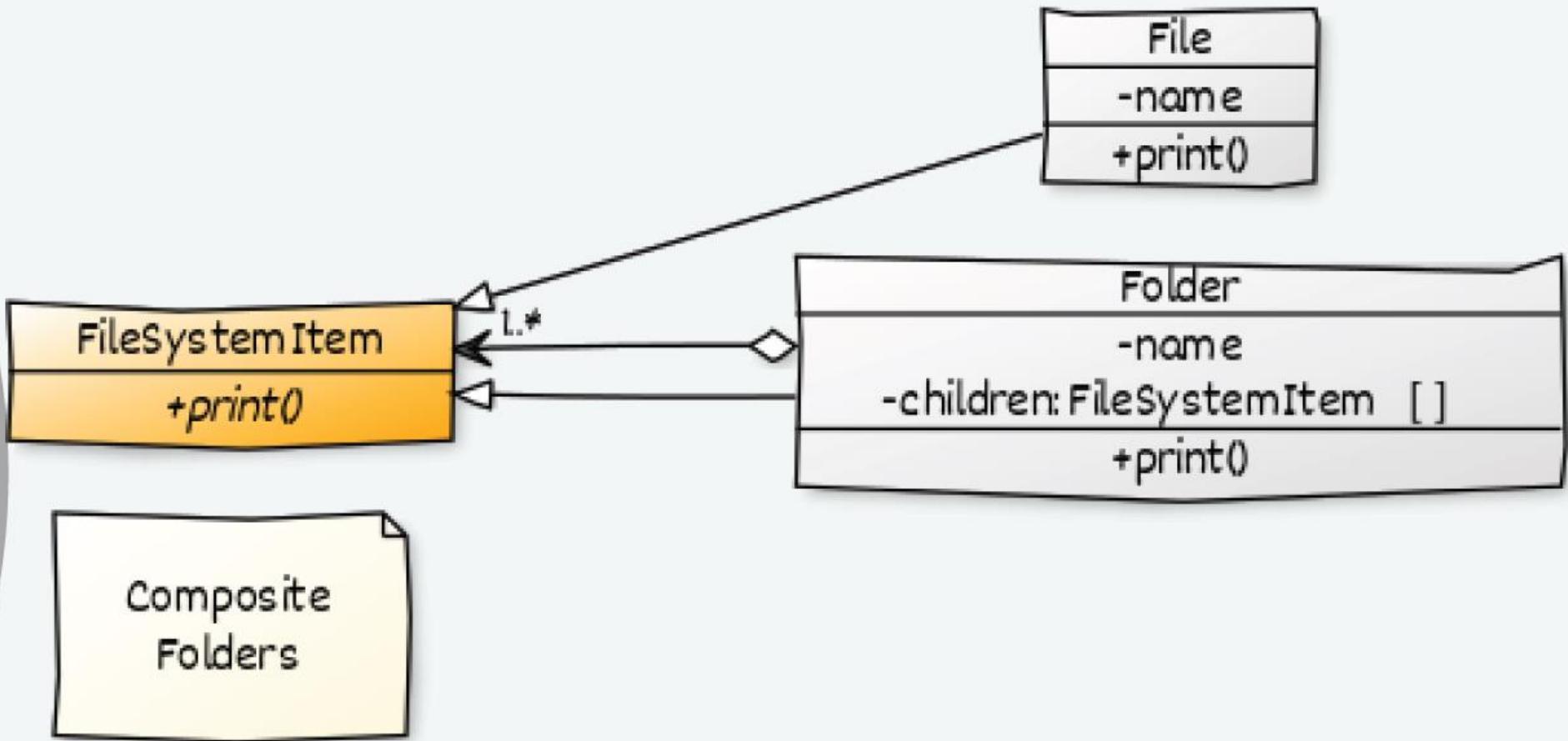




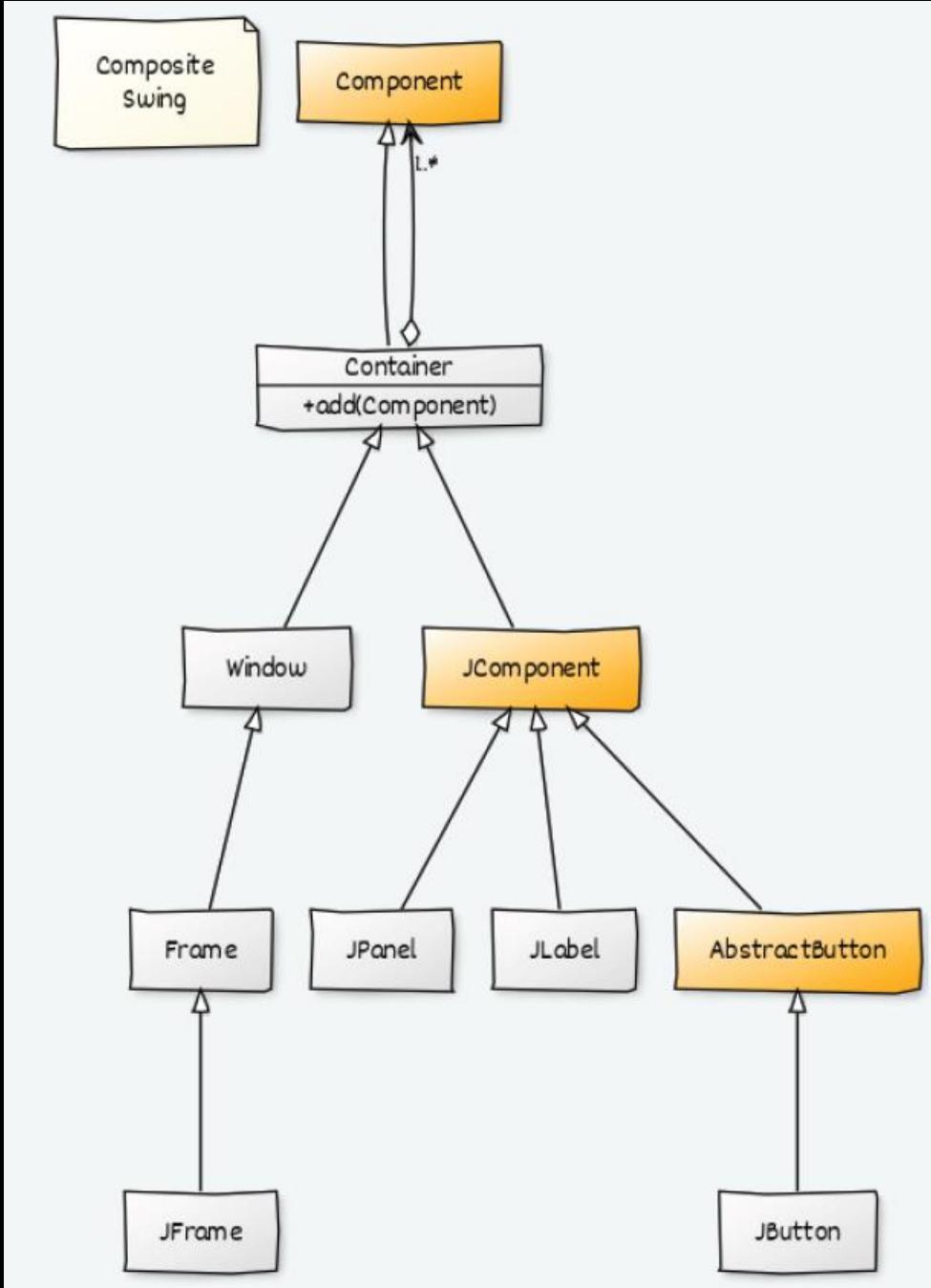
Compor objetos em estruturas de árvore para representar hierarquias todo-parte. Composite permite que clientes tratem objetos individuais e composições de objetos de maneira uniforme

GOF





CREATED WITH YUML



# DECORATOR

VAMOS ADICIONAR SÓ MAIS UM  
DETALHEZINHO?



# PROBLEMAS

- COMO POSSO ADICIONAR  
FUNCIONALIDADES DINAMICAMENTE EM  
UM OBJETO?



# SOLUÇÃO

- COM OBJETOS DECORATORS É POSSÍVEL ADICIONAR NOVAS RESPONSABILIDADES PARA UM OBJETO

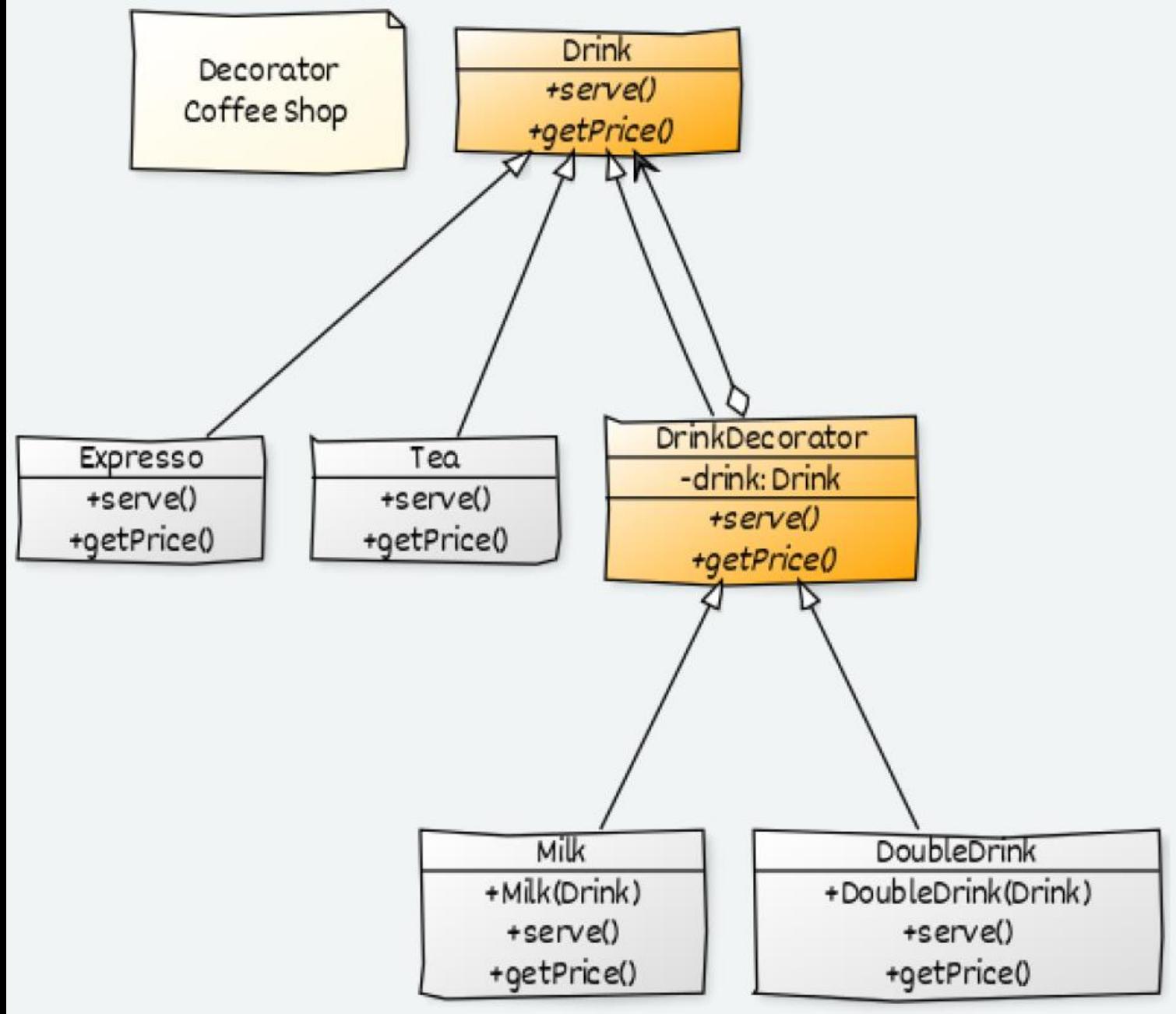


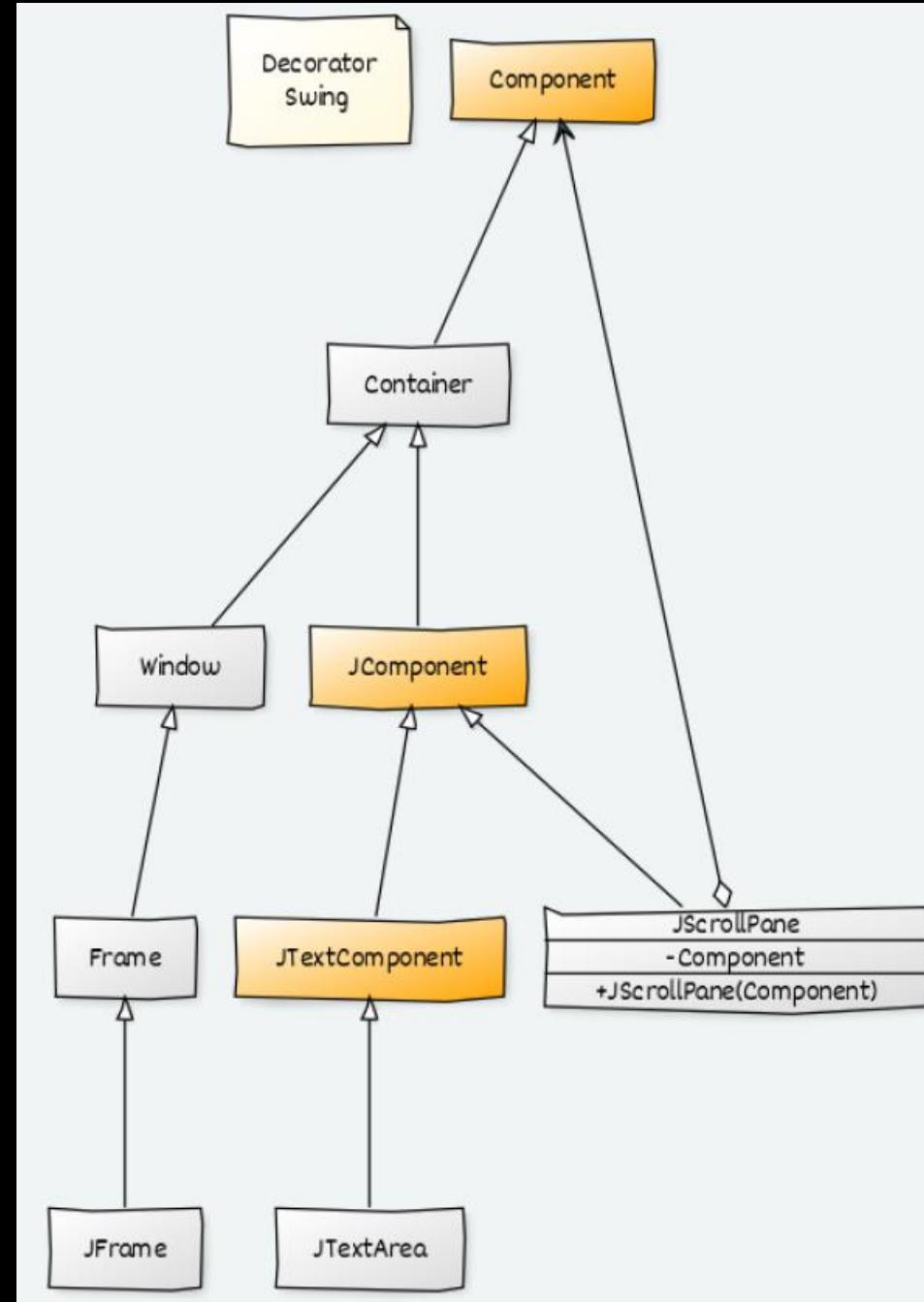


Anexar responsabilidades adicionais a um objeto dinamicamente. Decorators oferecem uma alternativa flexível ao uso de herança para estender uma funcionalidade.

**GOF**







# FACADE

AQUI ESTÁ O QUE VOCÊ PRECISA.  
SÓ NÃO REPARA A BAGUNÇA ;)



# PROBLEMAS

- COMO POSSO SIMPLIFICAR O ACESSO A UM SISTEMA COMPLEXO UTILIZANDO UMA INTERFACE SIMPLES?
- COMO REDUZIR A COMPLEXIDADE INTERNA DE UM SISTEMA PARA O CLIENTE?



# SOLUÇÃO

- DEFINIR UMA FACHADA QUE PROVISIONA UMA INTERFACE ÚNICA PARA UM CONJUNTO DE INTERFACES DE UM SISTEMA

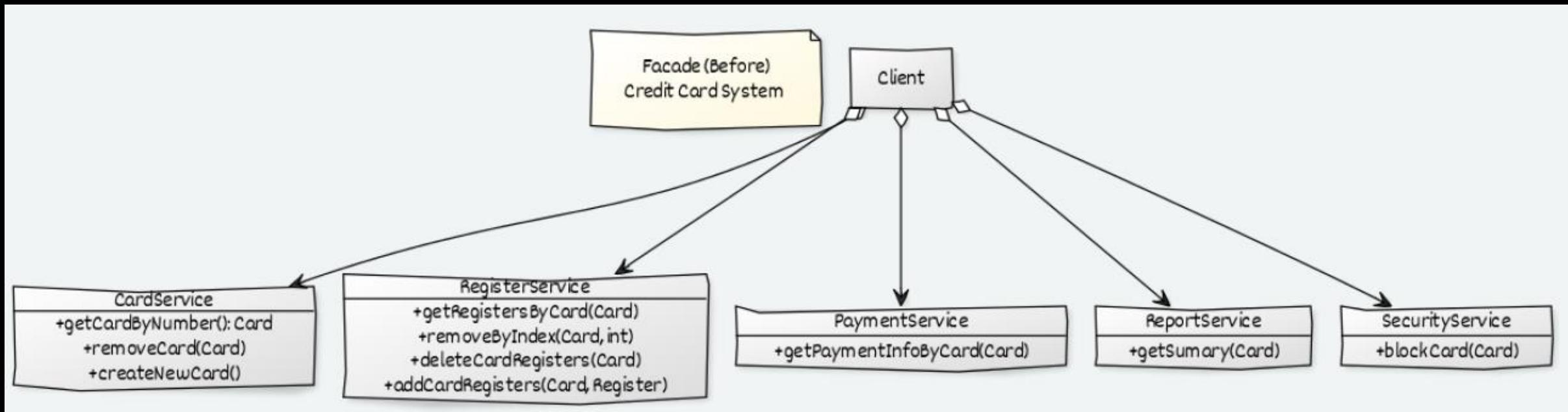


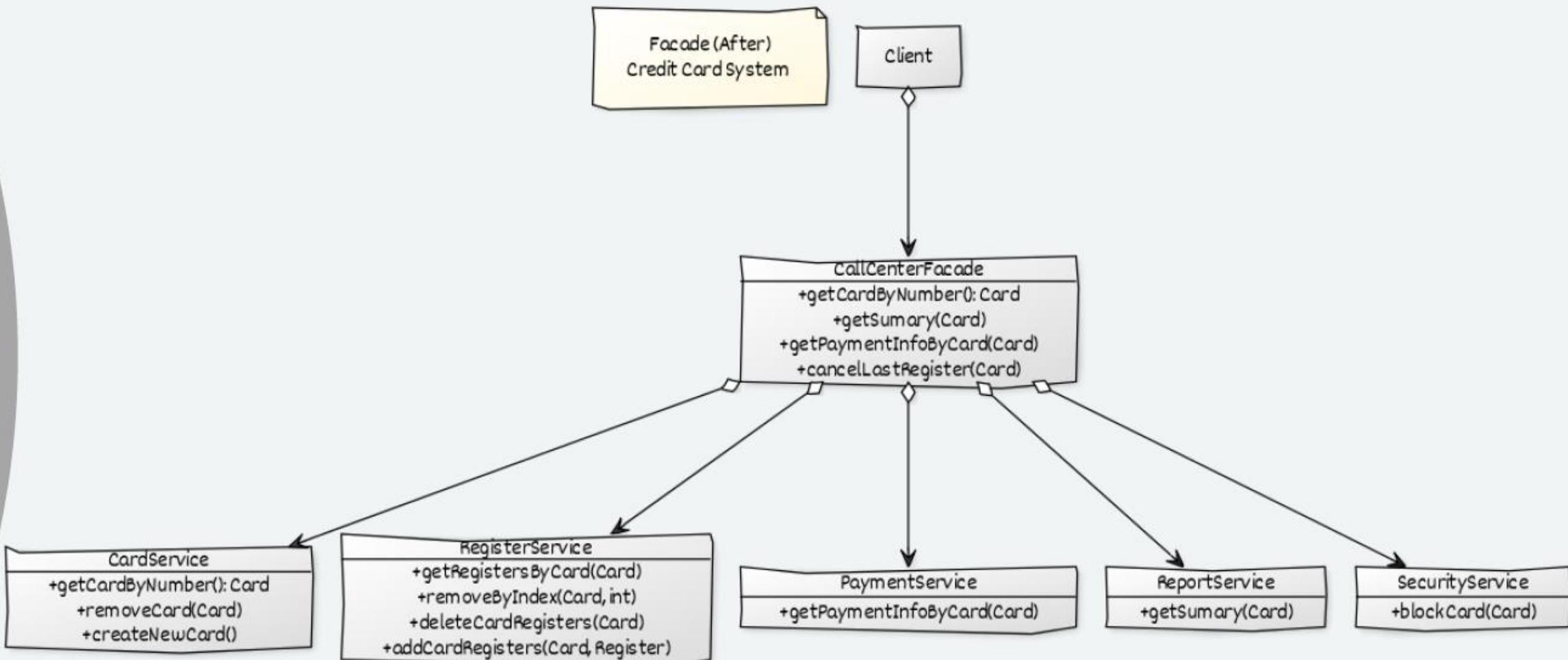


Oferecer uma interface única para um conjunto de interfaces de um subsistema. Façade define uma interface de nível mais elevado que torna o subsistema mais fácil de usar.

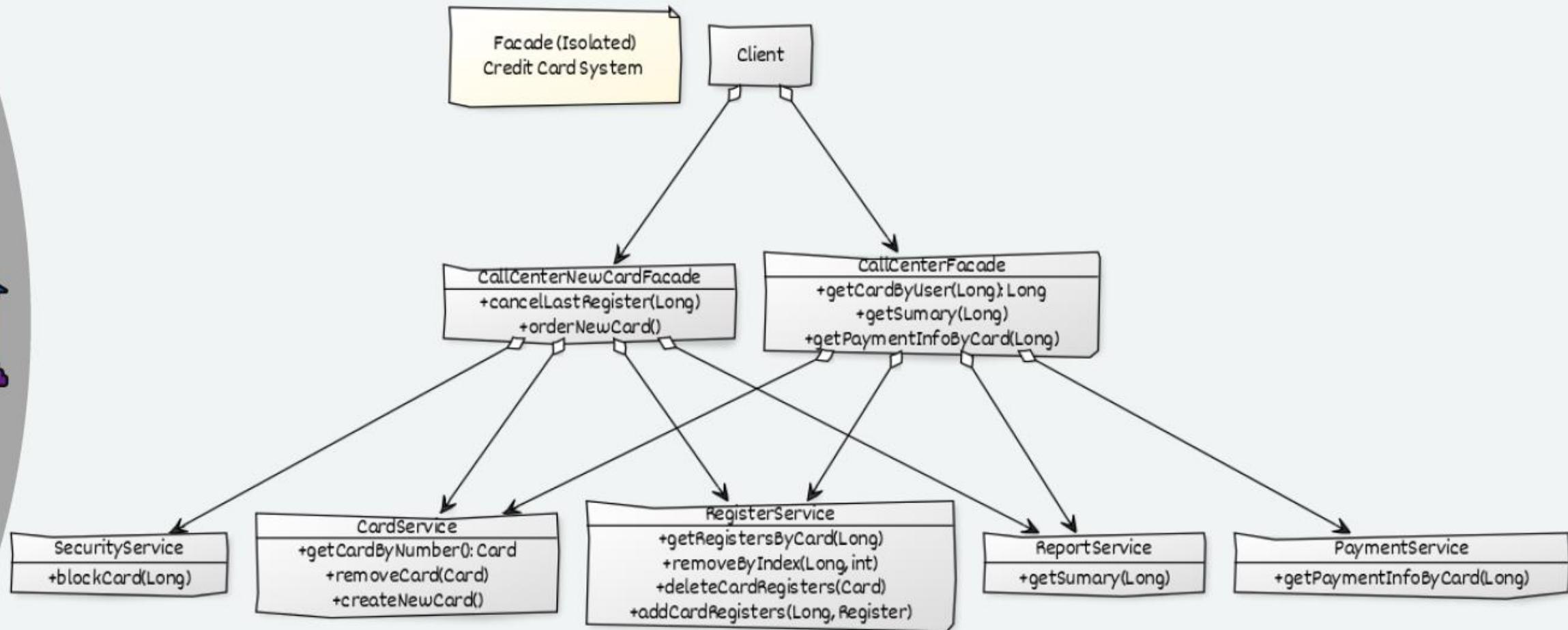
**GOF**







```
public class Client {  
  
    public static void main(String[] args) {  
        CallCenterFacade facade = new CallCenterFacade();  
  
        // Using system objects  
        Card card = facade.getCardByUser(123456L);  
        facade.getSummary(card);  
  
        //Using simple values  
        Long cardNumber = 123456L;  
        facade.getSummary(cardNumber);  
  
    }  
}
```



# FLYWEIGHT

É TIPO UM CACHE



# PROBLEMAS

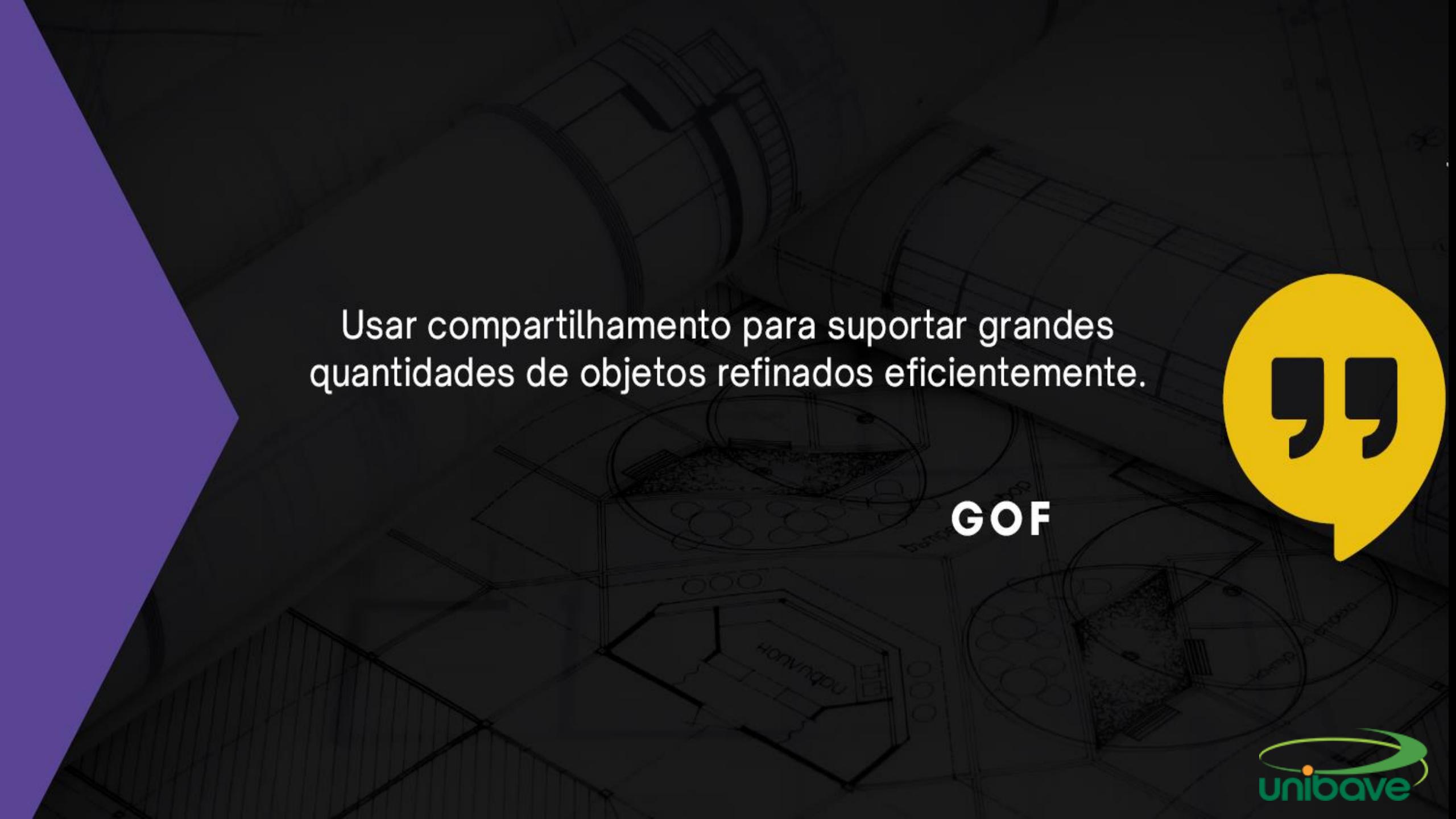
- COMO POSSO TRABALHAR COM UMA  
GRANDE QUANTIDADE DE OBJETOS NA  
MEMÓRIA DE UMA FORMA MAIS  
EFICIENTE?



# SOLUÇÃO

- DIVIDIR O OBJETO ENTRE VALORES INTRÍNSECOS E EXTRÍNSECOS...
- ... CRIAR UM FLYWEIGHT QUE ARMAZENE OS VALORES INTRÍNSECOS...
- ... OS CLIENTES IRÃO COMPARTILHAR ESTES FLYWEIGHTS, ADICIONANDO OS VALORES EXTRÍNSECOS PONTUALMENTE





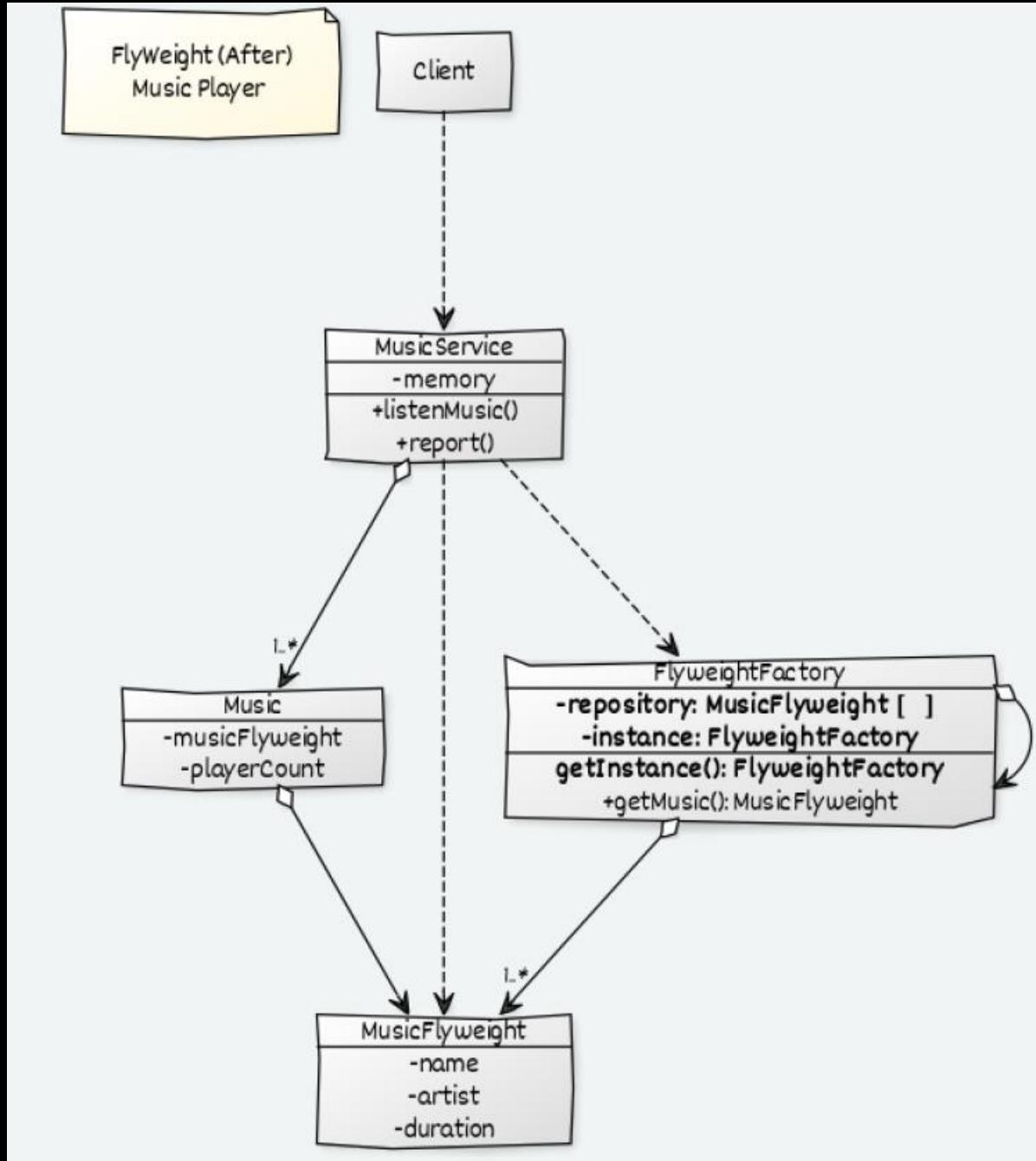
Usar compartilhamento para suportar grandes quantidades de objetos refinados eficientemente.

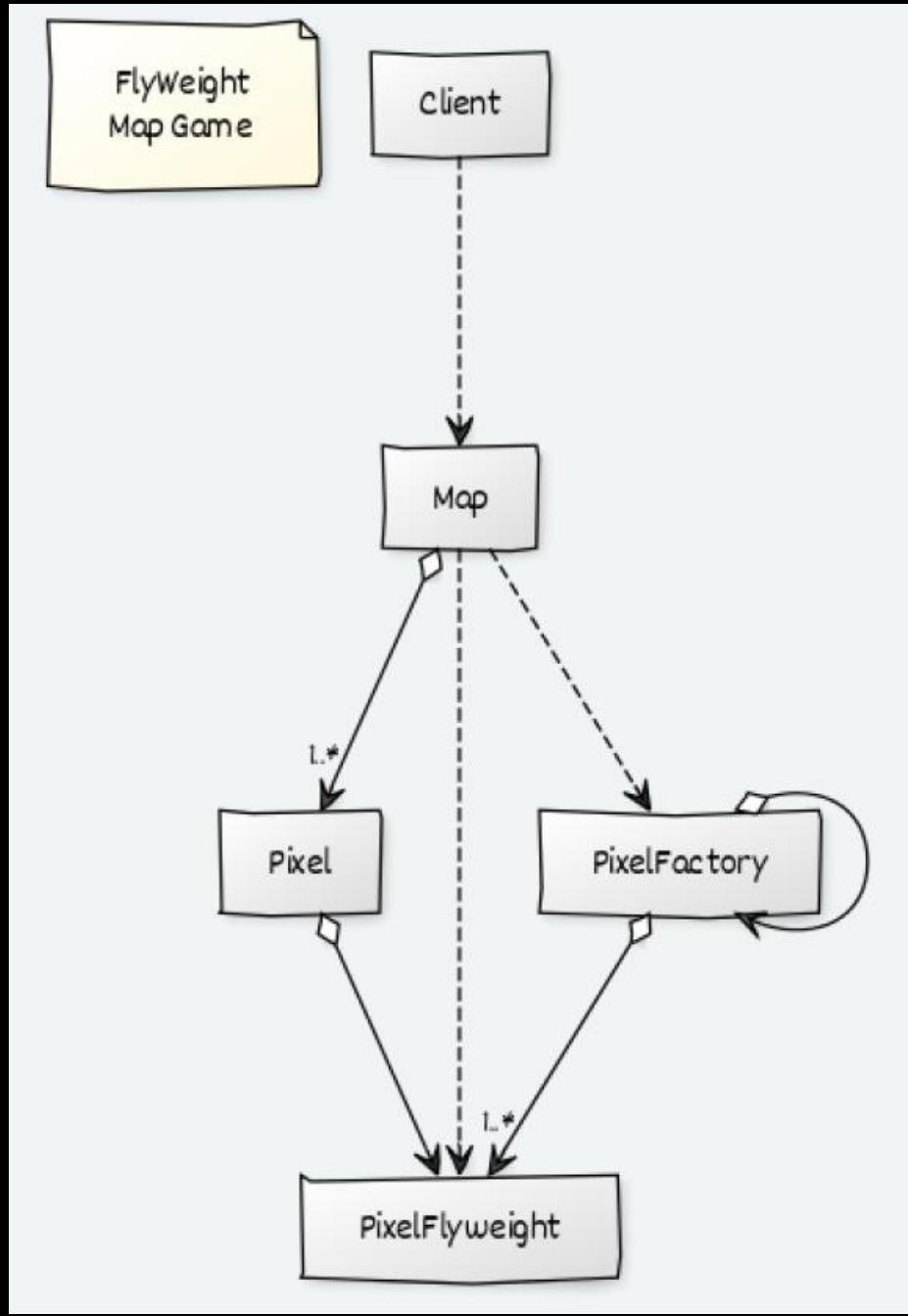
GOF





FlyWeight (Before)  
Music Player





# PROXY

EU SOU UM PROXY PARA O SEU  
CONHECIMENTO



# PROBLEMAS

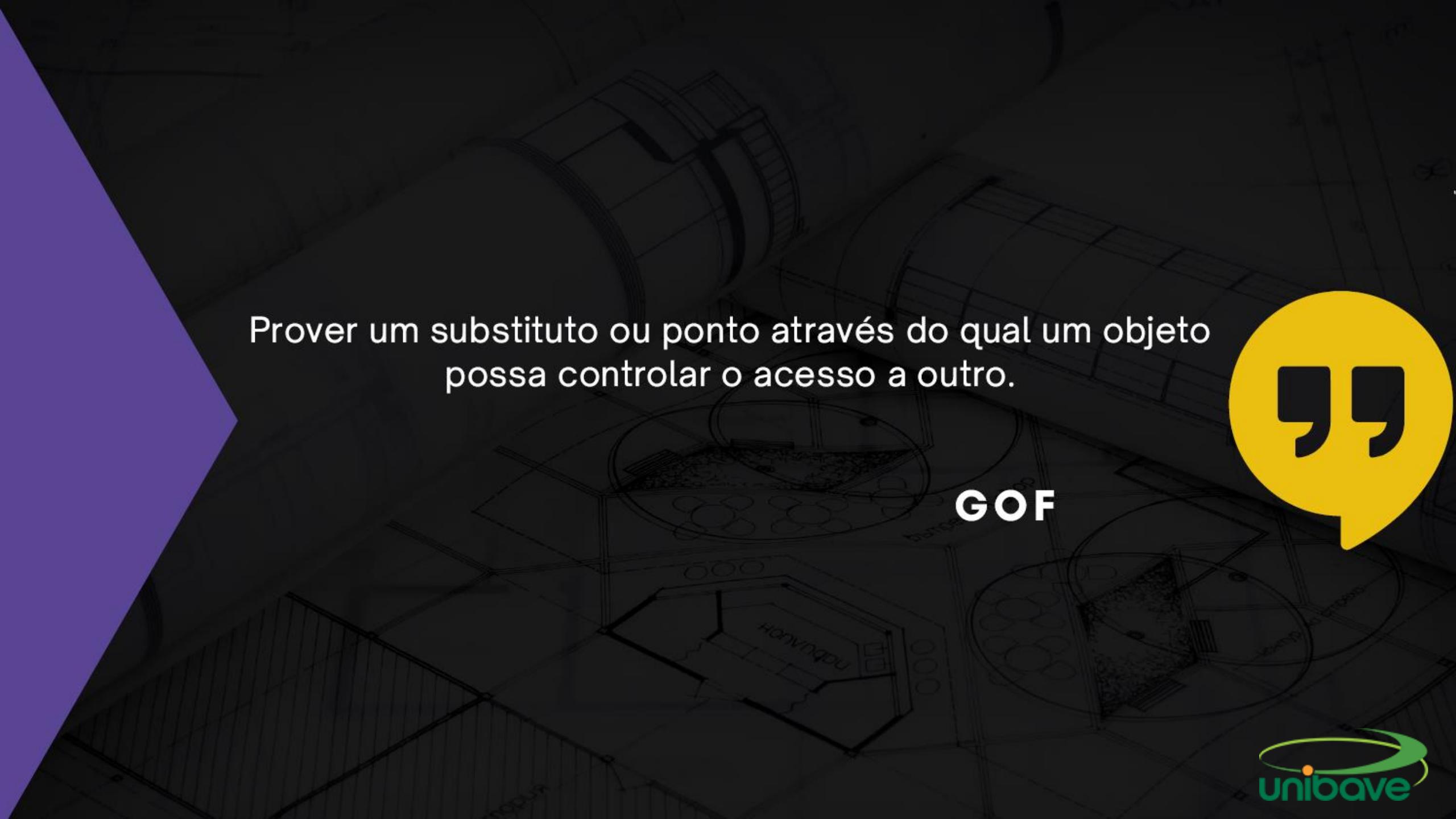
- COMO POSSO CONTROLAR O ACESSO A ALGUM OBJETO?
- COMO POSSO ADICIONAR FUNCIONALIDADES DURANTE O ACESSO A ALGUM OBJETO?



# SOLUÇÃO

- UTILIZAR UMA ESTRUTURA DE PROXY  
PARA ATUAR COMO SE FOSSE UM OUTRO  
OBJETO
- ... UTILIZANDO O PROXY COMO FORMA DE  
ACESSAR O OBJETO REAL, SERÁ POSSÍVEL  
CONTROLAR O SEU ACESSO

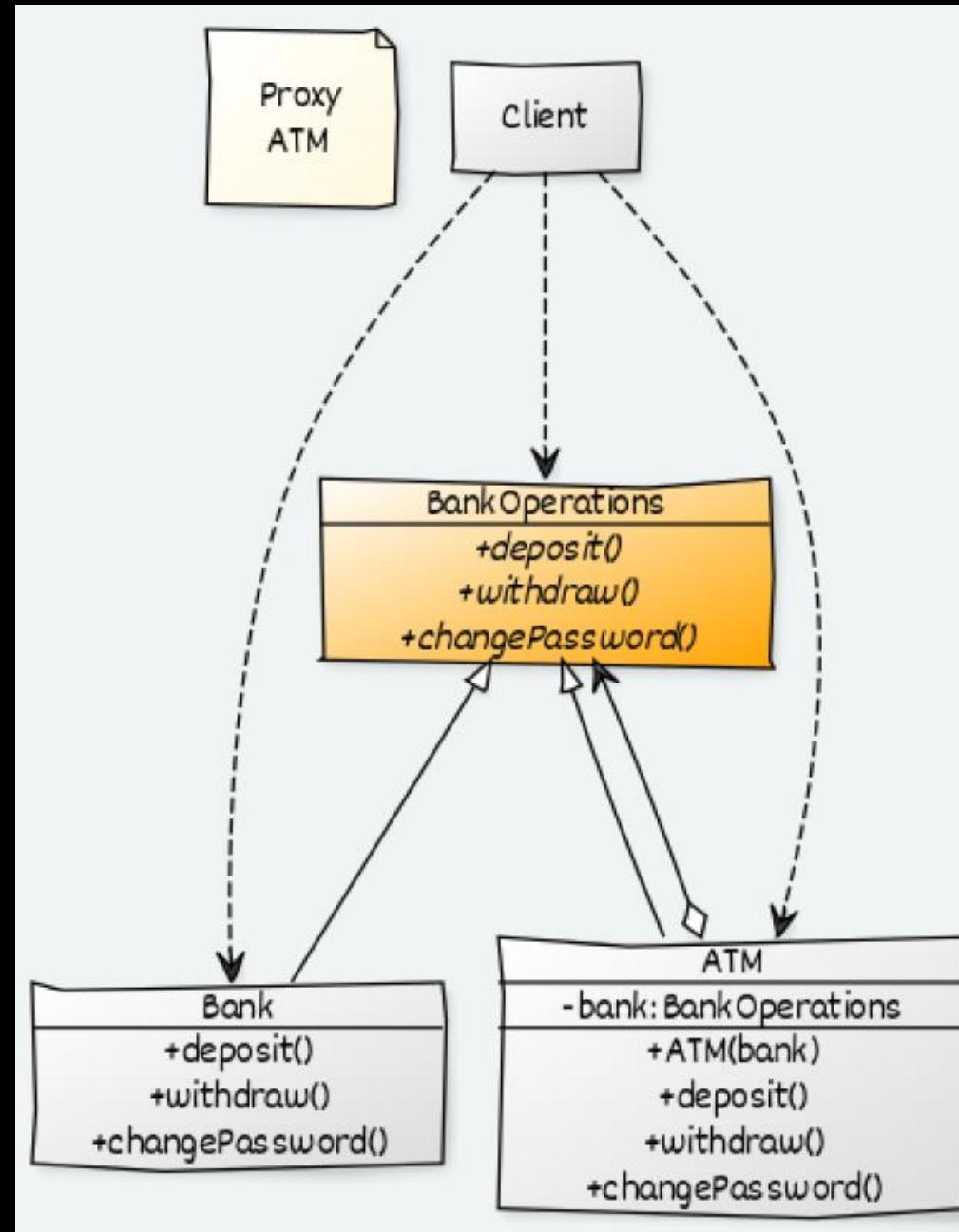


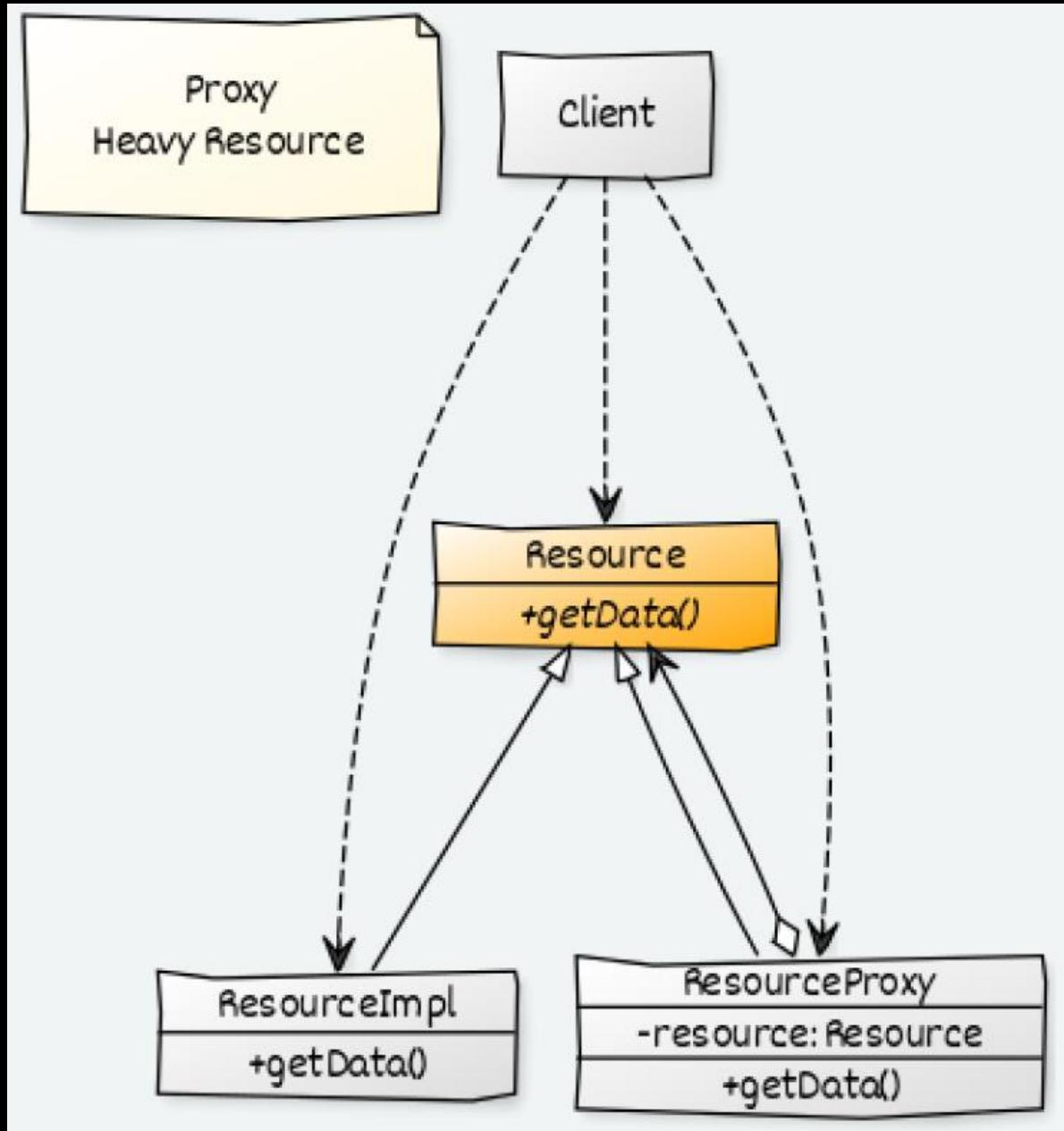


Prover um substituto ou ponto através do qual um objeto possa controlar o acesso a outro.

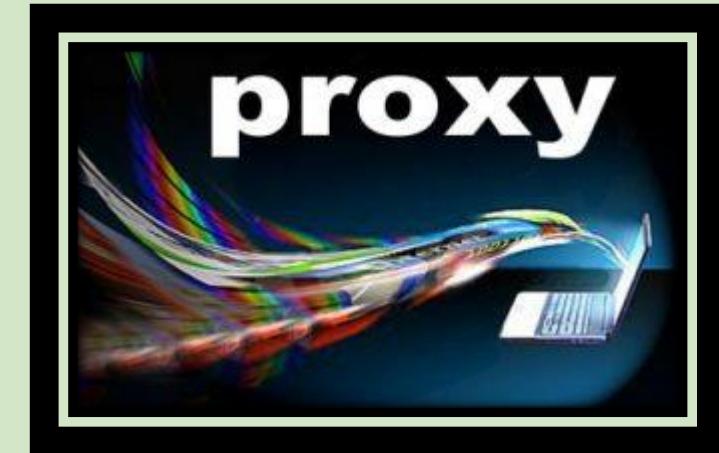
**GOF**







- Remote proxy
- Virtual proxy
- Protection proxy
- Smart Proxy



# MEDIATOR

APENAS UM GAROTO DE RECADOS



# PROBLEMAS

- COMO POSSO REDUZIR O ACOPLAMENTO ENTRE OBJETOS QUE PRECISAM SE COMUNICAR?
- COMO POSSO DEIXAR A INTERAÇÃO ENTRE OBJETOS MAIS DINÂMICA?



# SOLUÇÃO

- UTILIZAR UMA ESTRUTURA DE MEDIATOR PARA ENCAPSULAR A FORMA EM QUE OS OBJETOS SE COMUNICARÃO
- OS OBJETOS NÃO PRECISAM SE CONHECER, POIS TODA A INTERAÇÃO SERÁ REALIZADA ATRAVÉS DO MEDIATOR

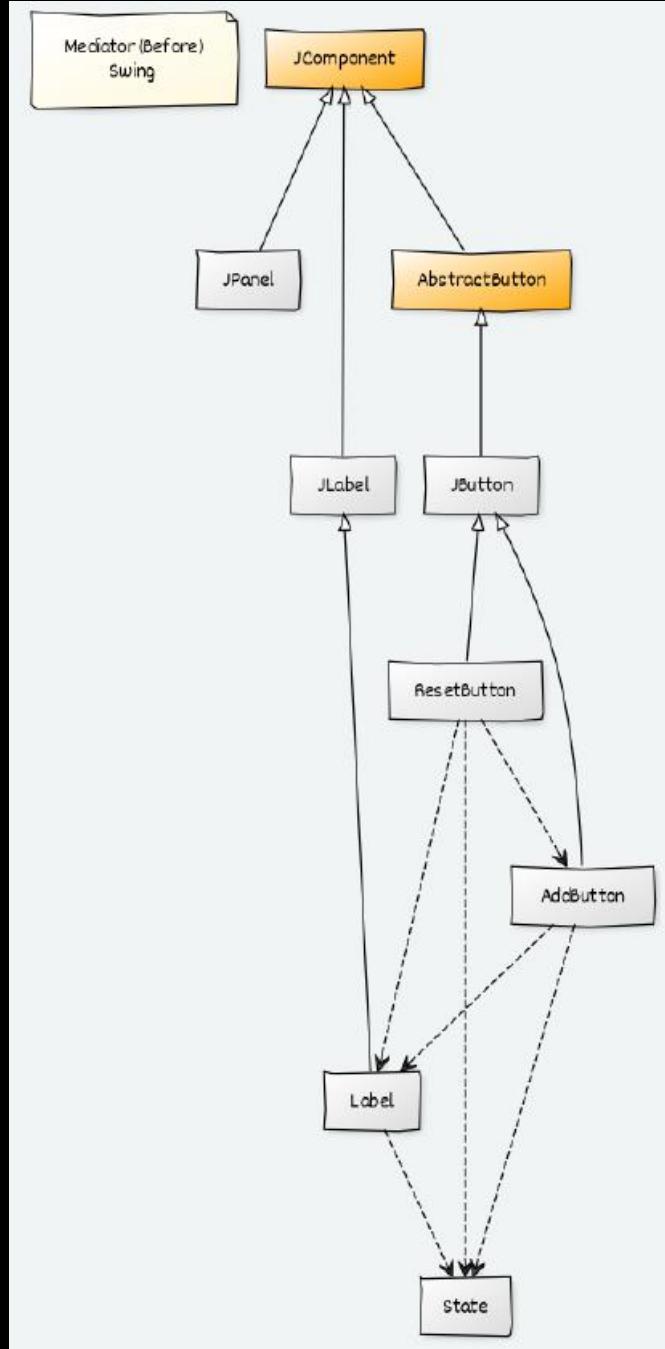


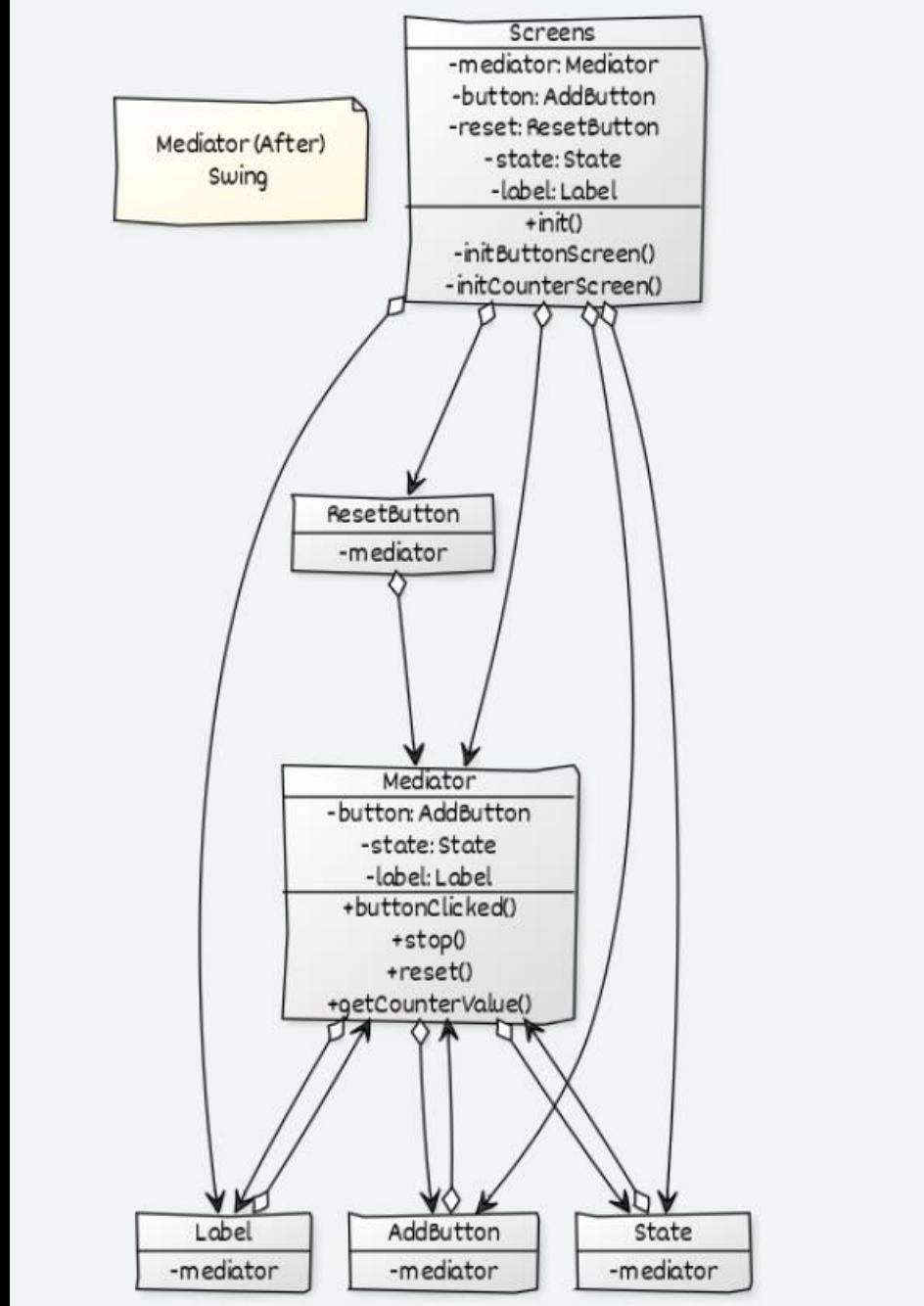


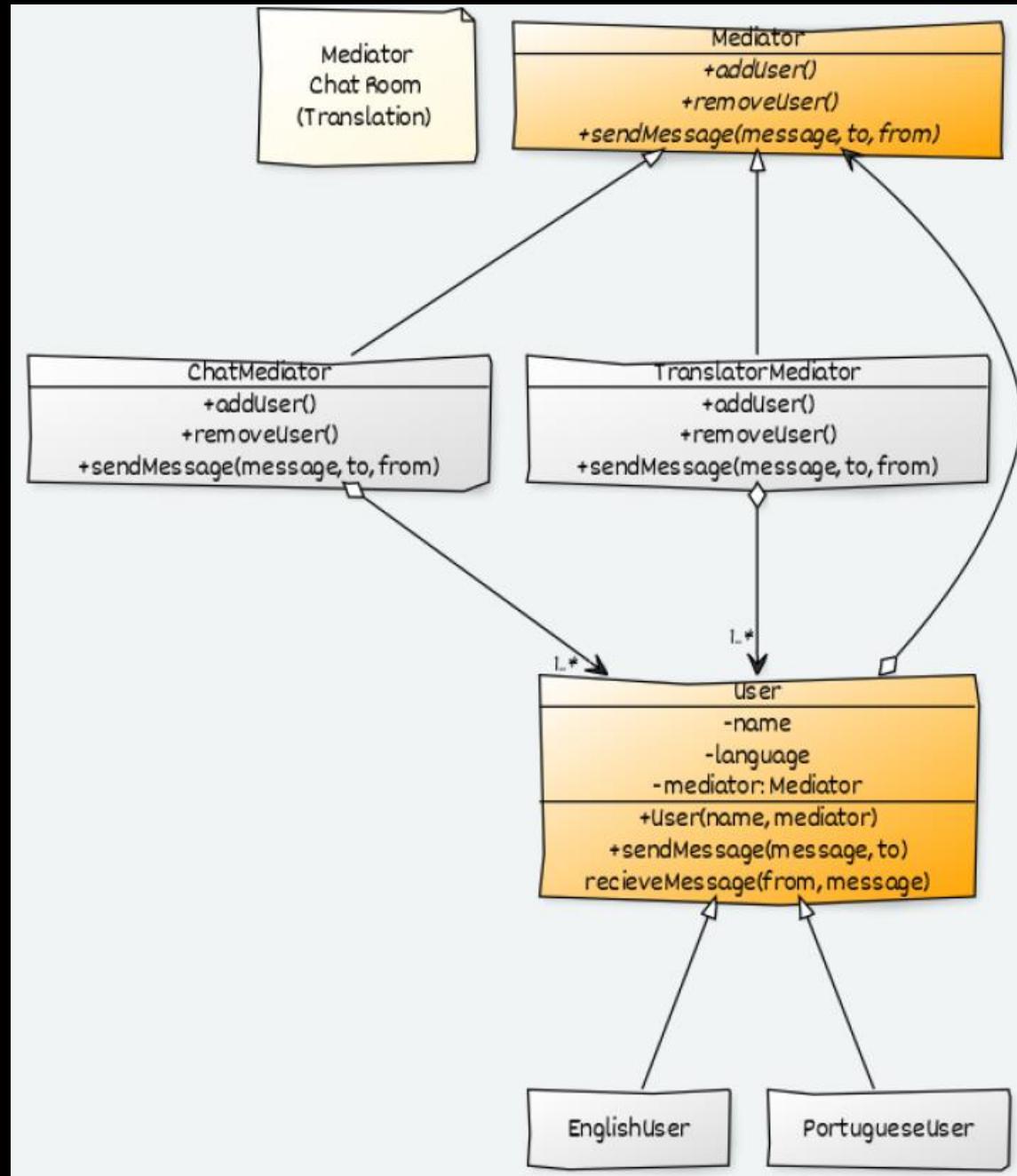
Definir um objeto que encapsula como um conjunto de objetos interagem. Mediator promove acoplamento fraco ao manter objetos que não se referem um ao outro explicitamente, permitindo variar sua interação independentemente.

GOF









# MÓDULO ← PADRÕES DE PROJETOS COMPORTAMENTAIS

VAMOS FOCAR AGORA EM COMO AS  
RESPONSABILIDADES SÃO PROPAGADAS



# CHAIN OF RESPONSABILITY

NINGUÉM SOLTA A MÃO DE NINGUÉM



# PROBLEMAS

- COMO POSSO EVITAR O ACOPLAMENTO ENTRE O REMETENTE DE UMA SOLICITAÇÃO E SEU RECEPTOR?
- COMO PERMITIR QUE MAIS QUE UM OBJETO POSSA ATENDER ALGUMA REQUISIÇÃO?



# SOLUÇÃO

- DEFINIR UMA CADEIA DE OBJETOS ONDE CADA UM PODERÁ RESPONDER ÀQUELA SOLICITAÇÃO OU ENVIAR PARA O PRÓXIMO OBJETO TRATÁ-LA.
- QUEM FAZ A SOLICITAÇÃO NÃO PRECISA SABER O TAMANHO DA CADEIA, OU MESMO COMO (POR QUEM) ELA SERÁ RESOLVIDA

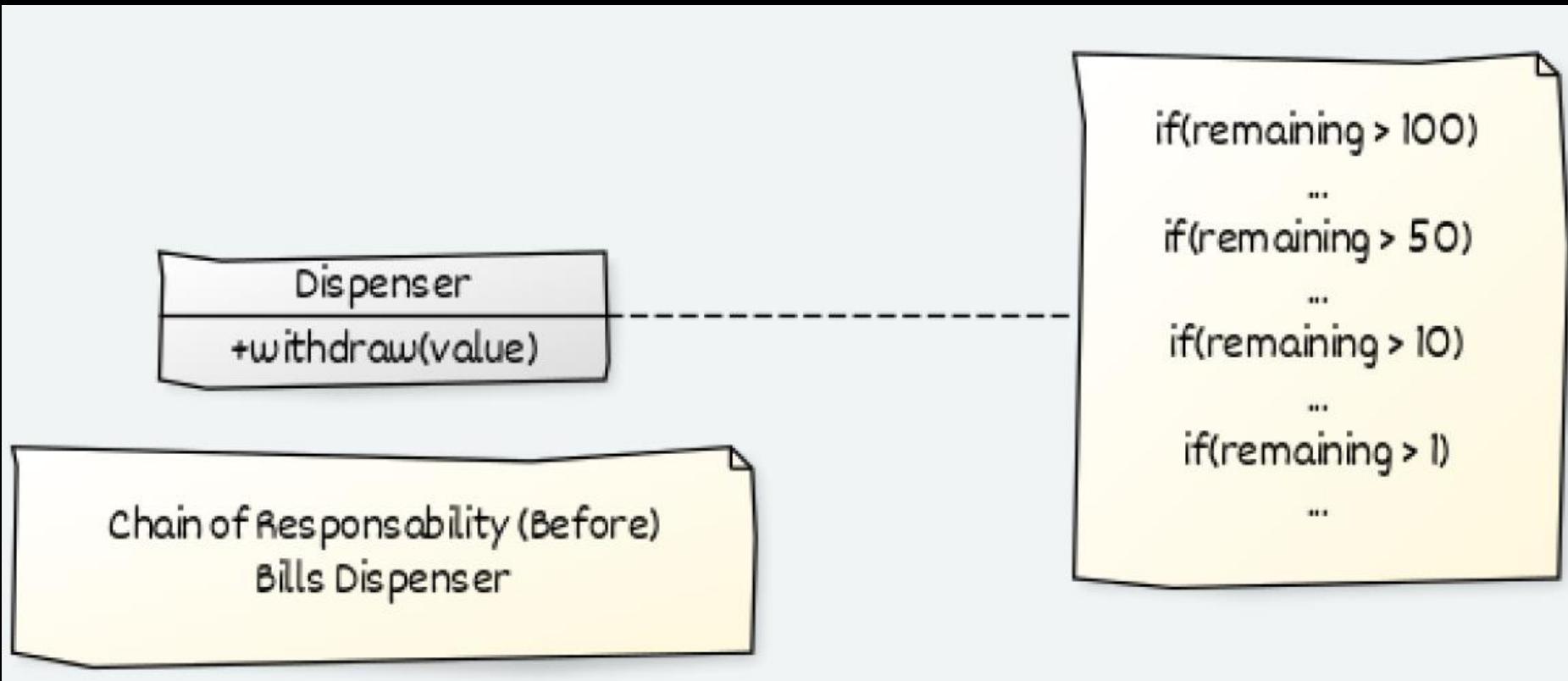


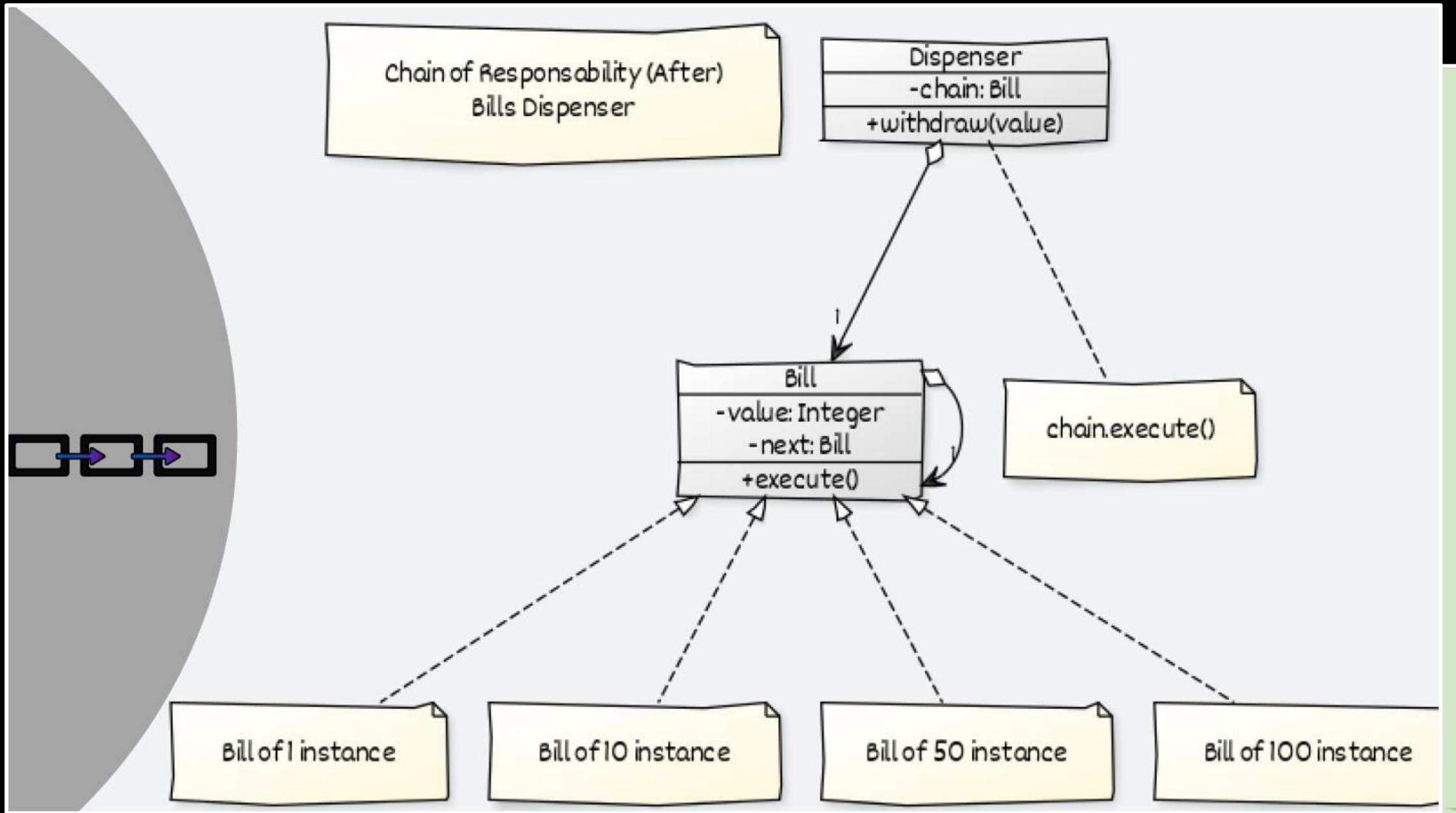


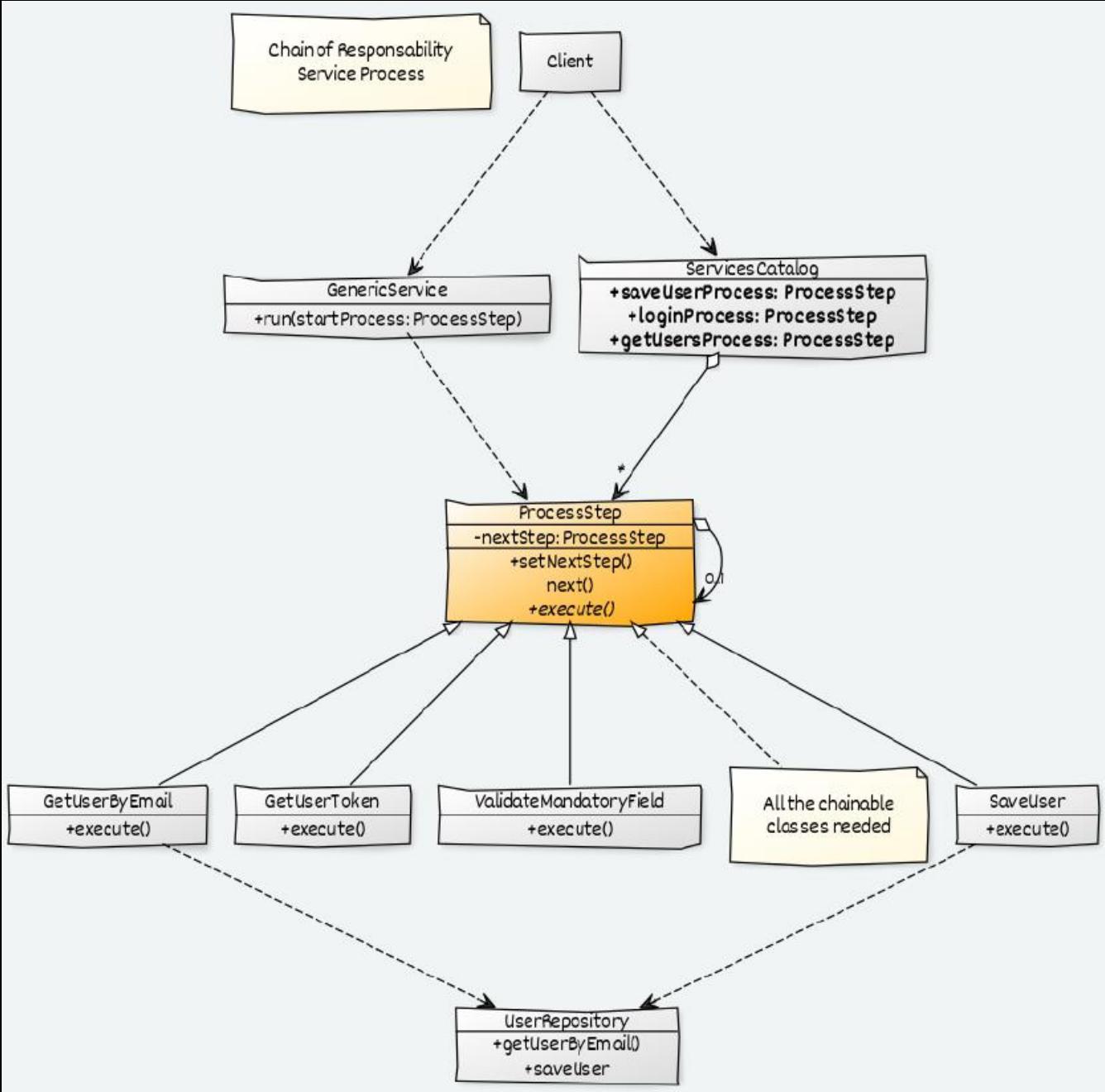
Evita acoplar o remetente de uma requisição ao seu destinatário ao dar a mais de um objeto a chance de servir a requisição. Compõe os objetos em cascata e passa a requisição pela corrente até que um objeto a sirva.



**GOF**









# MEMENTO

CHECKPOINT!



# PROBLEMAS

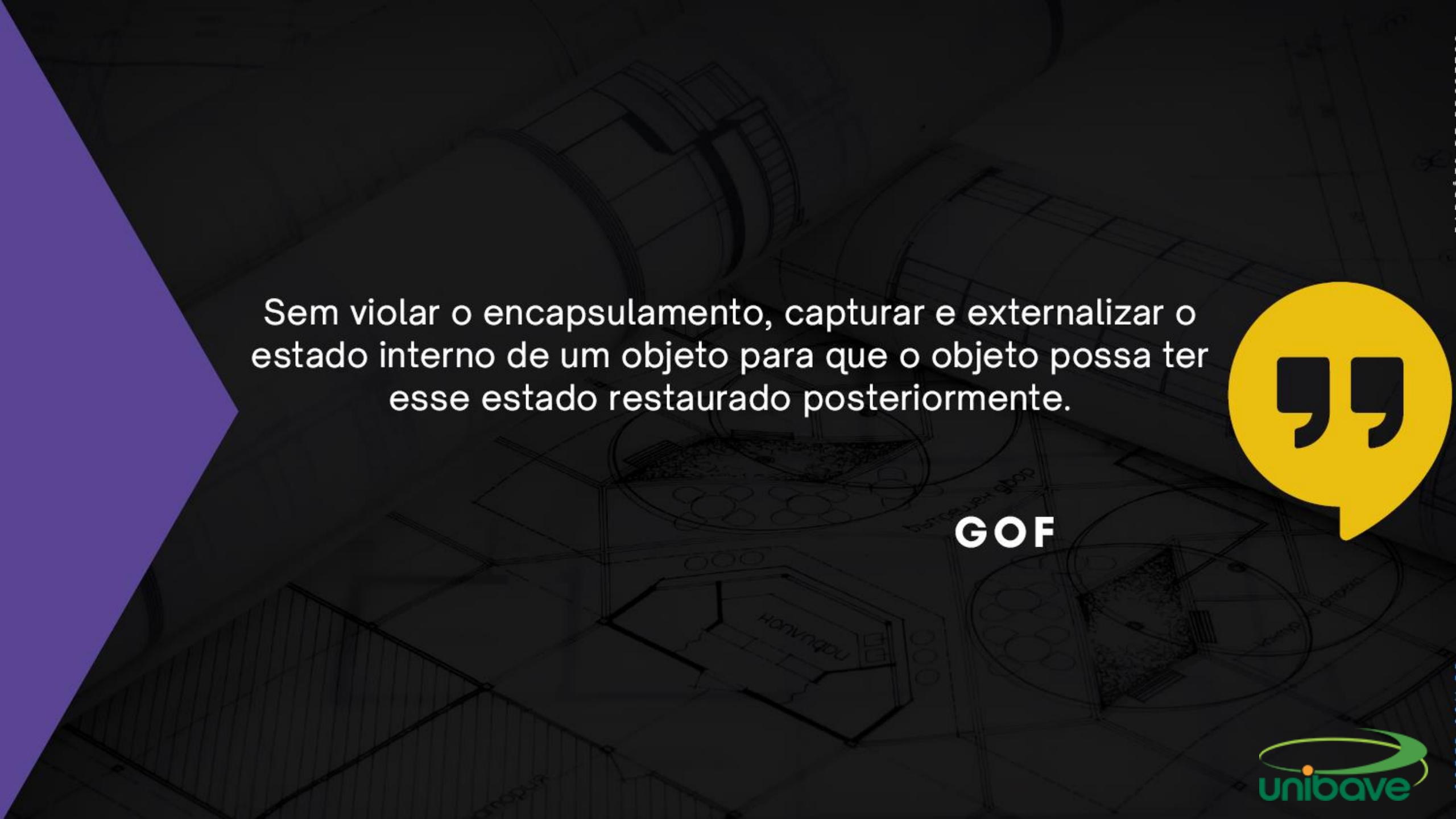
- COMO UM OBJETO PODE CAPTURAR SEU ESTADO INTERNO PARA PODER SER RESTAURADO POSTERIORMENTE?



# SOLUÇÃO

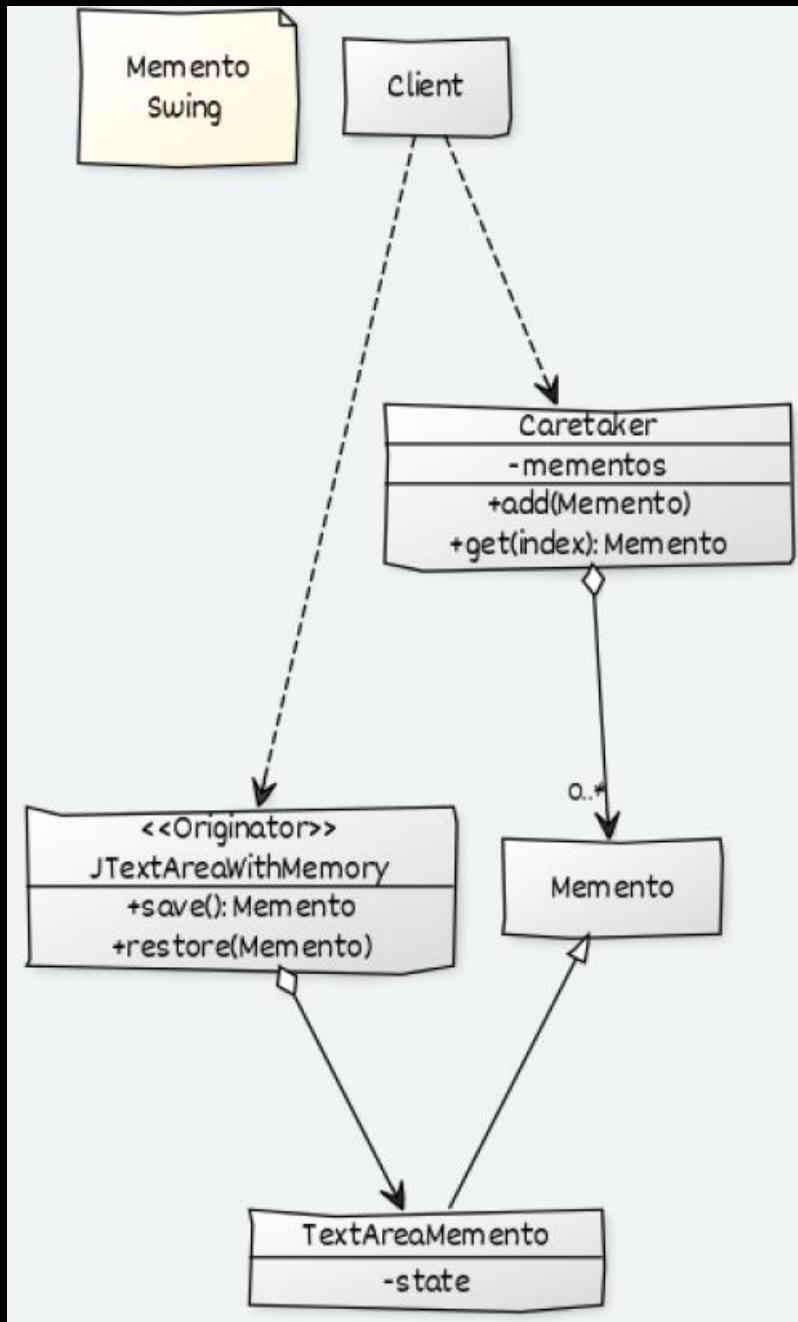
- DEFINIR UMA ESTRUTURA DE MEMENTO  
ONDE CADA ESTADO PODERÁ SER SALVO  
E RESTAURADO QUANDO NECESSÁRIO





Sem violar o encapsulamento, capturar e externalizar o estado interno de um objeto para que o objeto possa ter esse estado restaurado posteriormente.

GOF





**COMMAND**

JUST DO IT!



# PROBLEMAS

- COMO POSSO REPRESENTAR UMA REQUISIÇÃO DENTRO DE UM OBJETO?
- PRECISO FAZER UMA REQUISIÇÃO MAS NÃO SEI COMO ELA SERÁ RESOLVIDA OU ATÉ MESMO QUEM IRÁ RESPONDÊ-LA?



# SOLUÇÃO

- ENCAPSULAR A REQUISIÇÃO EM UM OBJETO COMMAND SEPARADO
- O COMANDO NÃO TEM OS DETALHES DE QUEM E COMO SERÁ RESOLVIDO

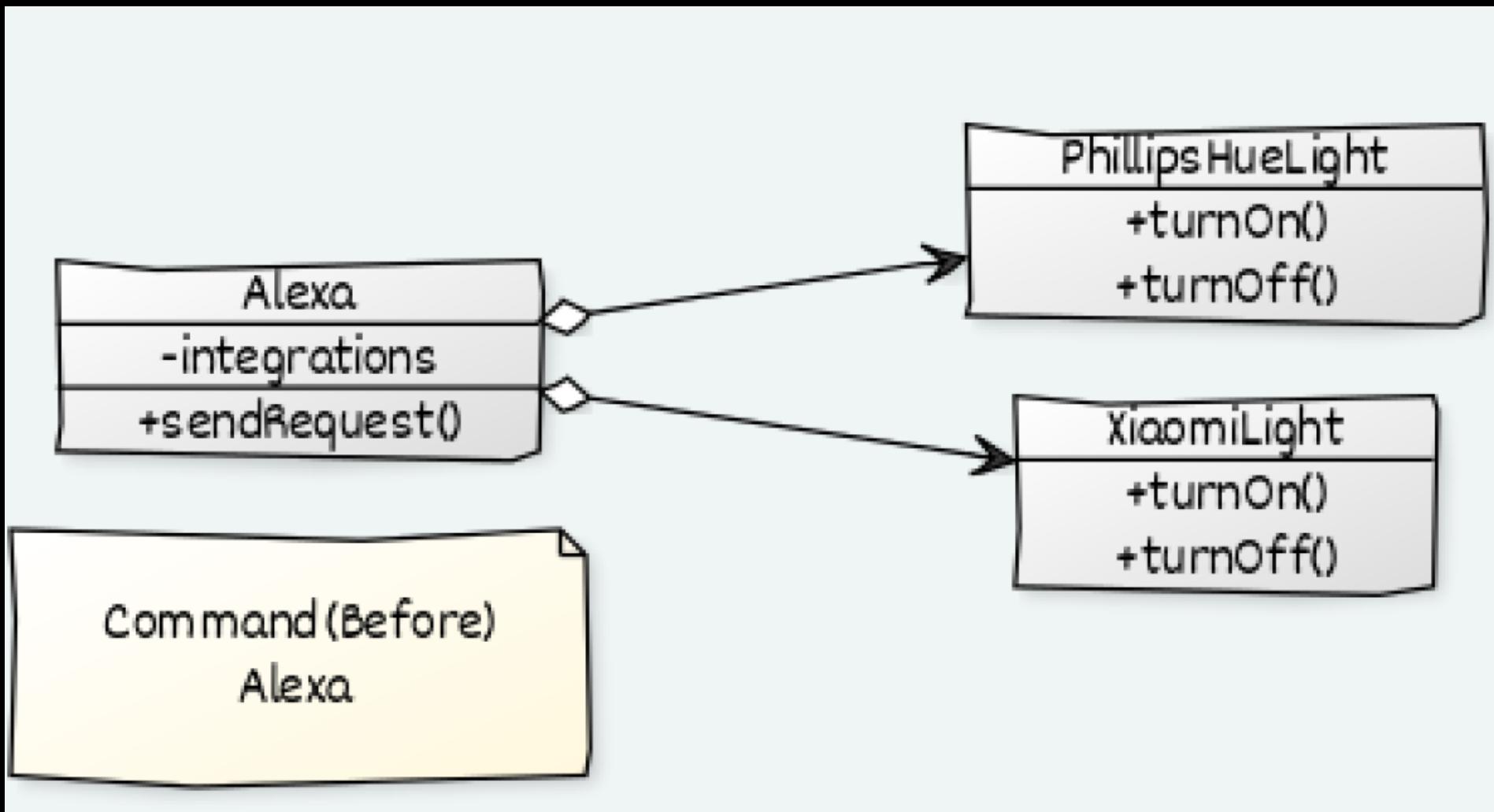


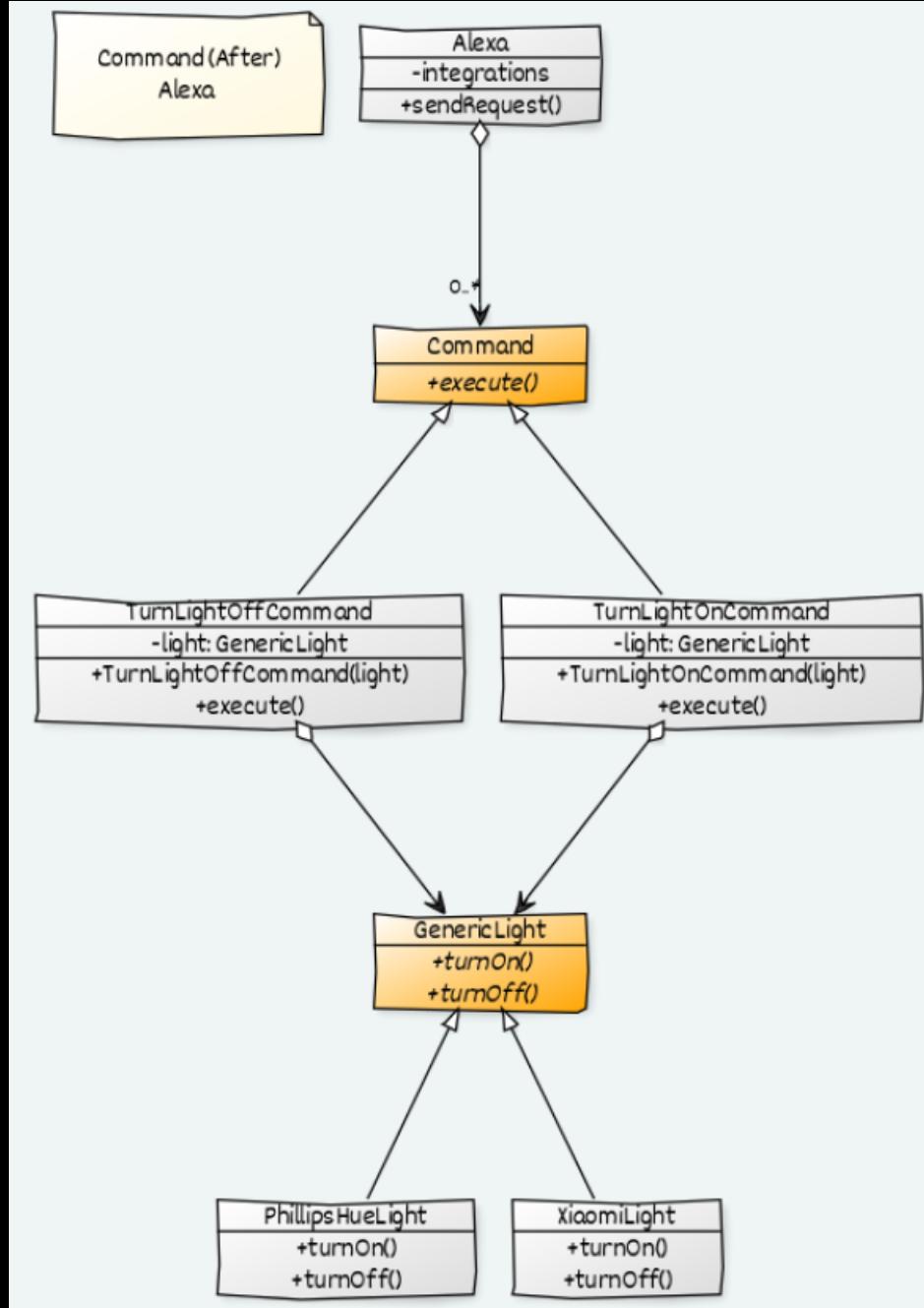


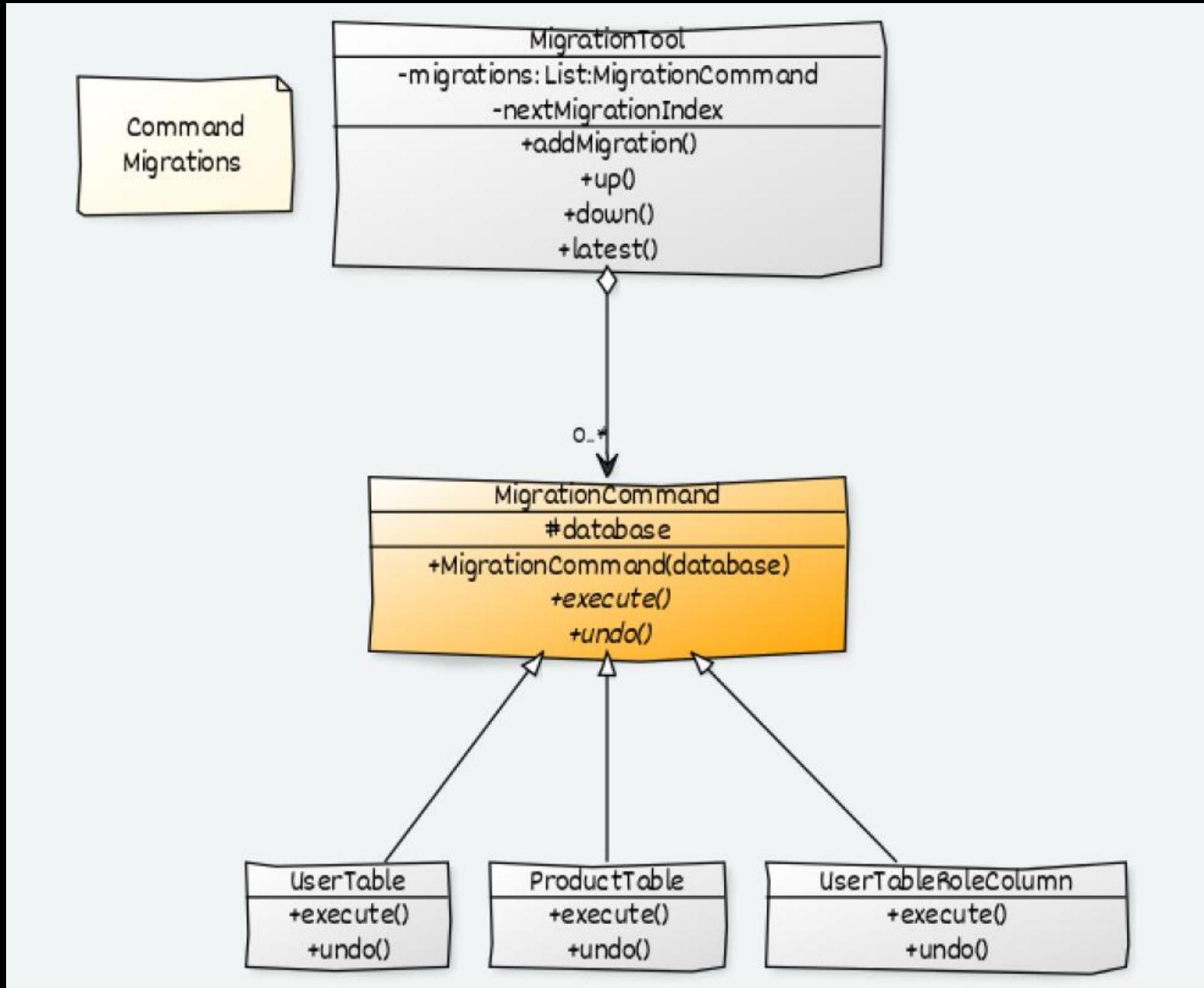
Encapsular uma requisição como um objeto, permitindo que clientes parametrizem diferentes requisições, filas ou requisições de log, e suportar operações reversíveis.

**GOF**



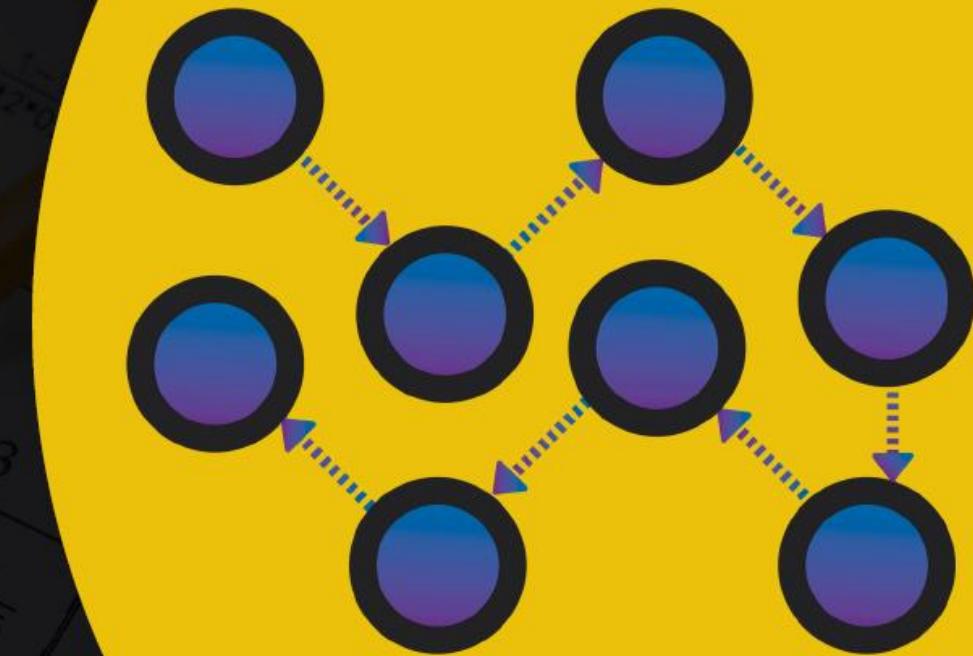






# ITERATOR

PRIMEIRO ELE, DEPOIS ELE ALÍ E DEPOIS  
AQUELE LÁ...



# PROBLEMAS

- COMO POSSO NAVEGAR ATRAVÉS DE  
UMA COLEÇÃO DE OBJETOS SEM A  
NECESSIDADE DE CONHECER OS  
DETALHES DESTA ESTRUTURA?



# SOLUÇÃO

- ENCAPSULAR A LÓGICA DE NAVEGAÇÃO  
ENTRE OS ELEMENTOS DESTA COLEÇÃO  
EM UMA ESTRUTURA DE ITERATOR

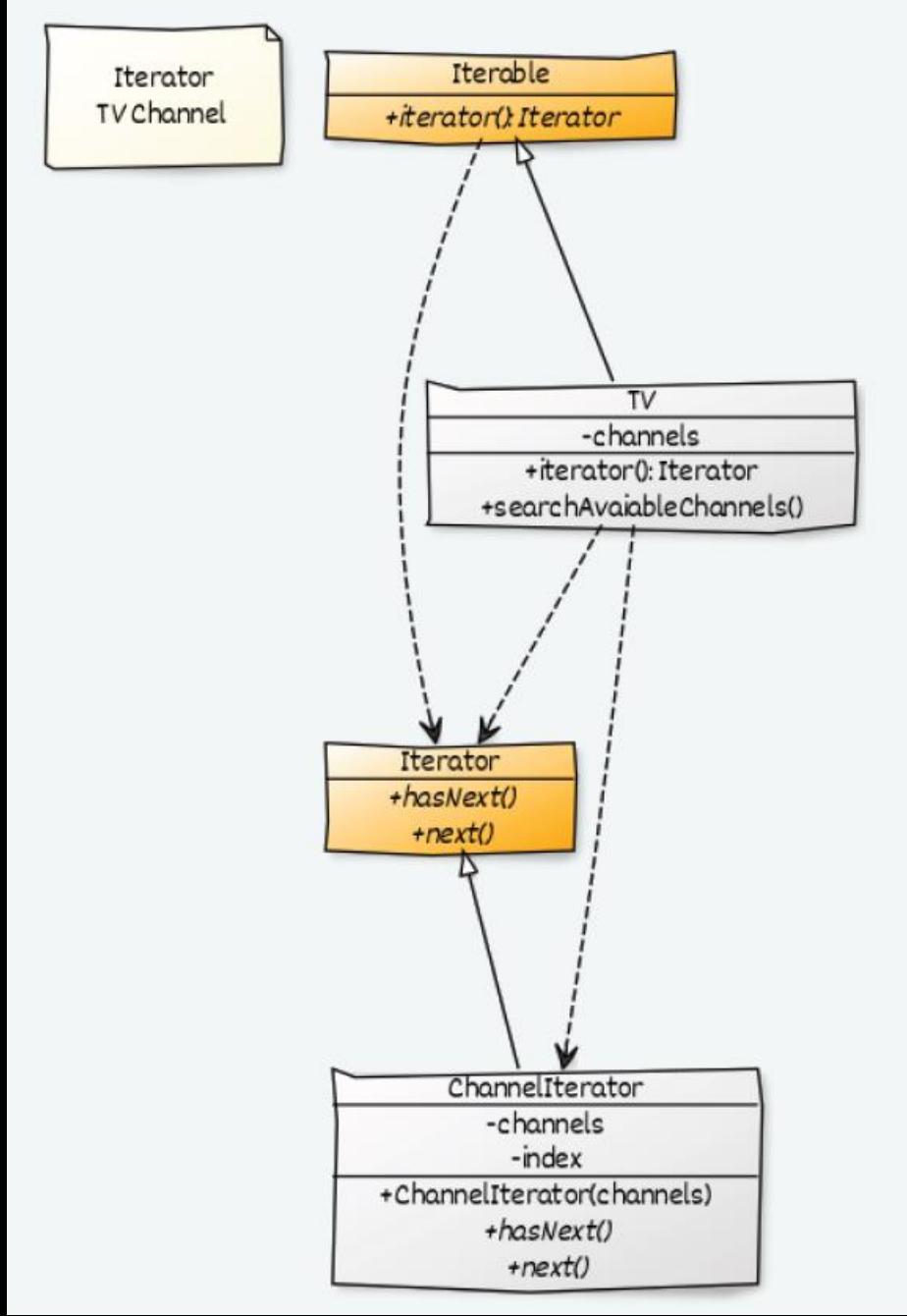




Prover uma maneira de acessar os elementos de um objeto agregado seqüencialmente sem expor sua representação interna.

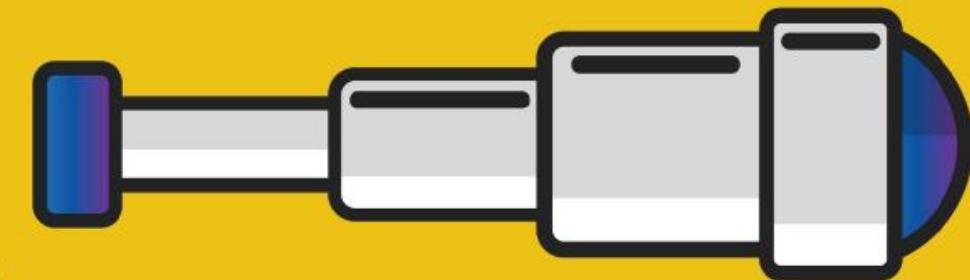
**GOF**





# OBSERVER

DON'T CALL ME, I'LL CALL YOU!



# PROBLEMAS

- COMO POSSO MODELAR UM RELACIONAMENTO 1-N SEM DEIXAR TODOS ELES ACOPLADOS?
- COMO UM OBJETO PODE NOTIFICAR OUTROS OBJETOS QUANDO NECESSÁRIO?



# SOLUÇÃO

- CRIAR UMA ESTRUTURA DE OBSERVER  
PARA QUE ELE POSSA NOTIFICAR TODOS  
OS OBJETOS QUE SOLICITARAM SER  
AVISADOS QUANDO UM DETERMINADO  
EVENTO OCORRA

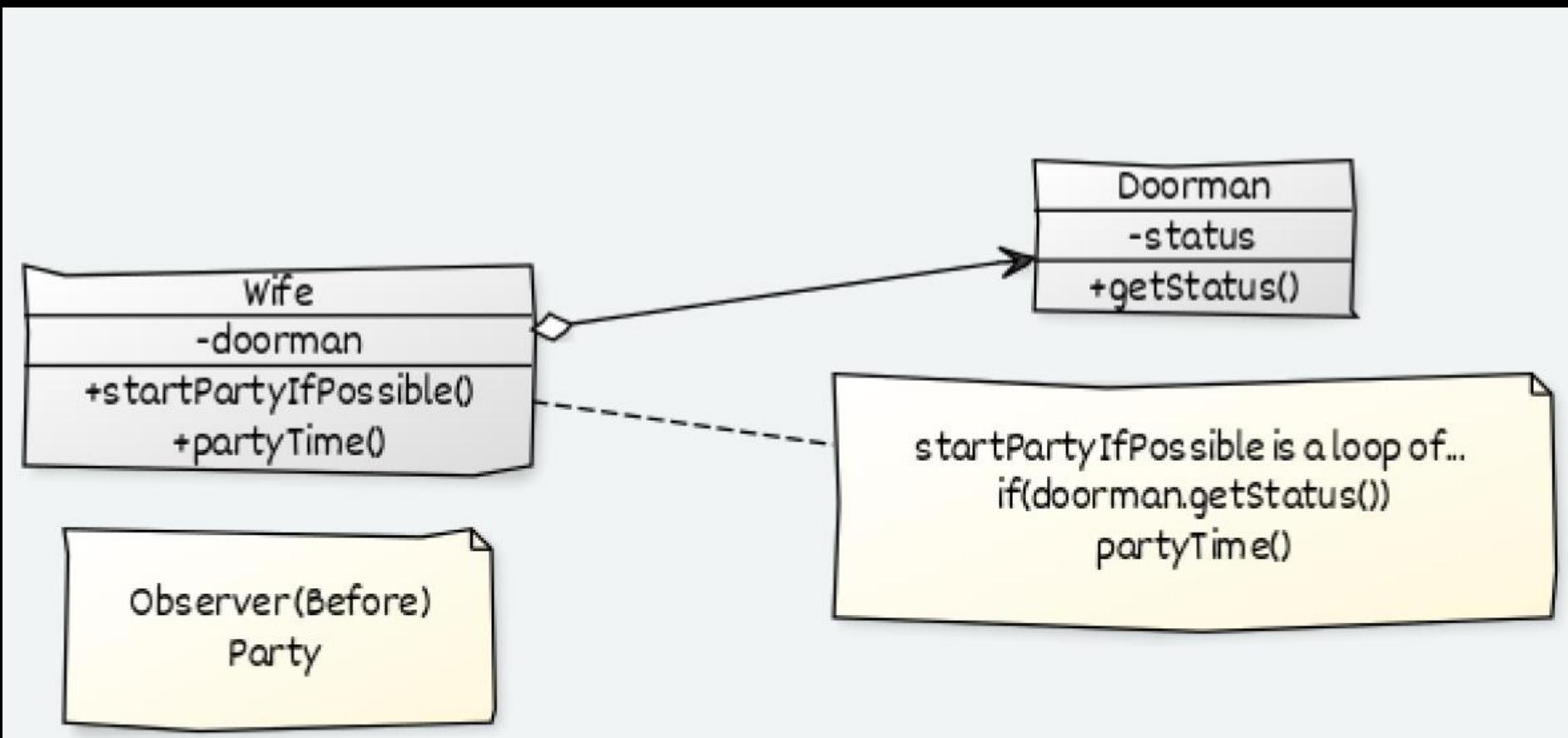


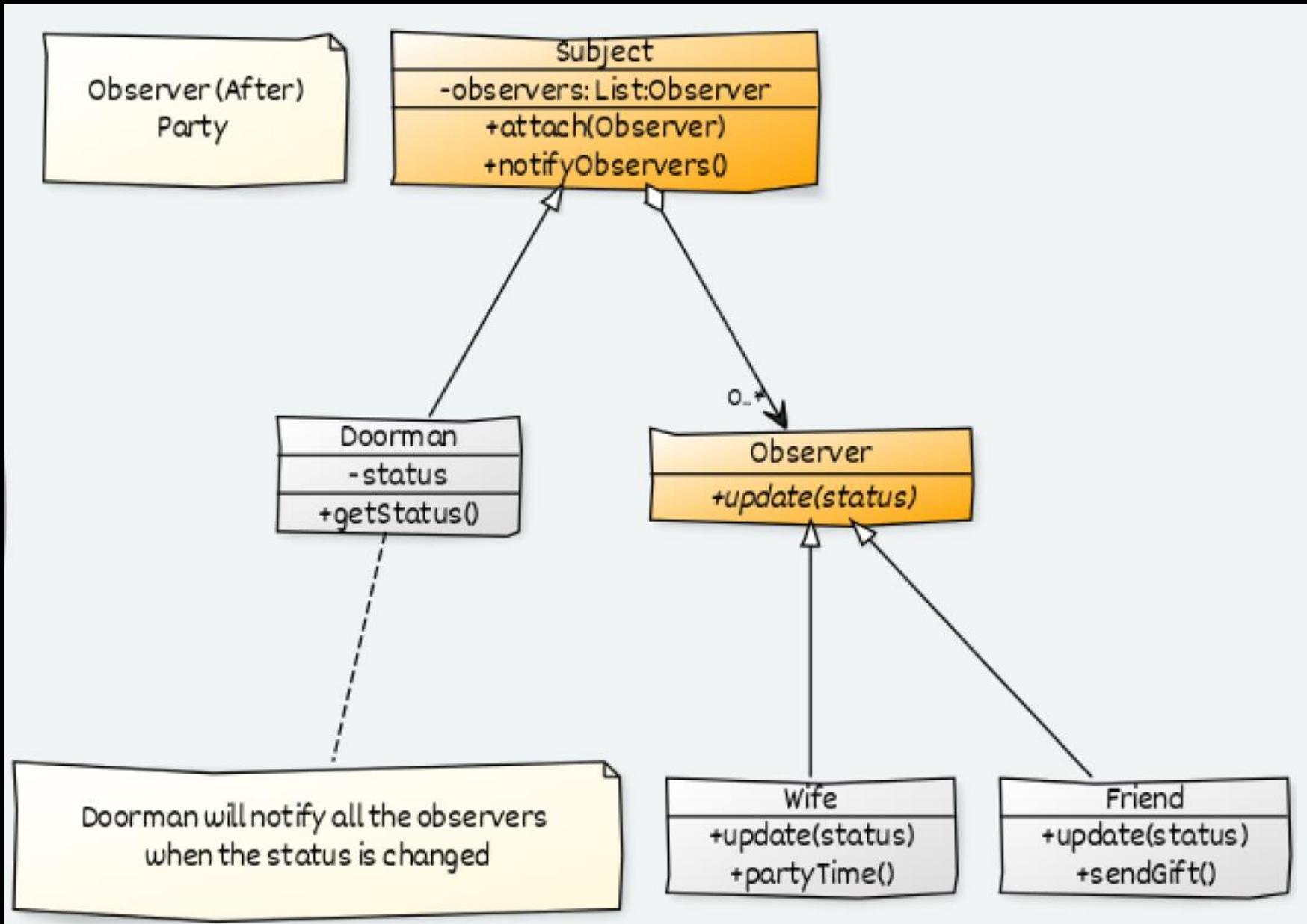


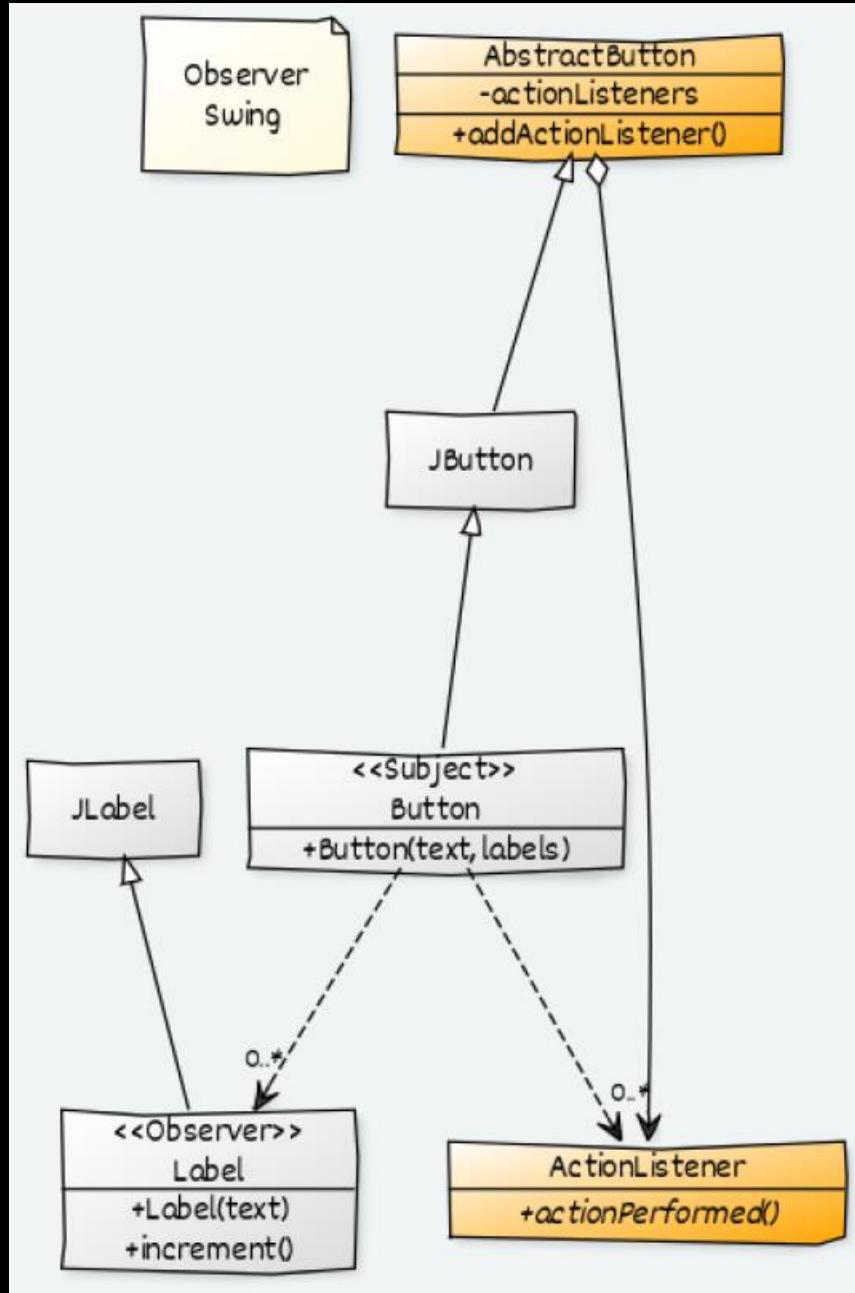
Definir uma dependência um-para-muitos entre objetos para que quando um objeto mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente.

GOF











**STATE**

A MÁQUINA DE TURING



# PROBLEMAS

- COMO POSSO ALTERAR O  
COMPORTAMENTO DE UM OBJETO  
QUANDO SEU ESTADO INTERNO MUDA?
- COMO PERMITIR QUE NOVOS  
COMPORTAMENTOS SEJAM ADICIONADOS  
E INTEGRADOS COM OS DEMAIS?



# SOLUÇÃO

- MODELAR OS COMPORTAMENTOS POSSÍVEIS ATRAVÉS DE STATES
- DEFINIR COMO SERÃO REALIZADAS AS MUDANÇAS DE ESTADOS
- CADA STATE IRÁ TOMAR CONTROLE DA EXECUÇÃO DE ACORDO COM O ESTADO INTERNO DO OBJETO



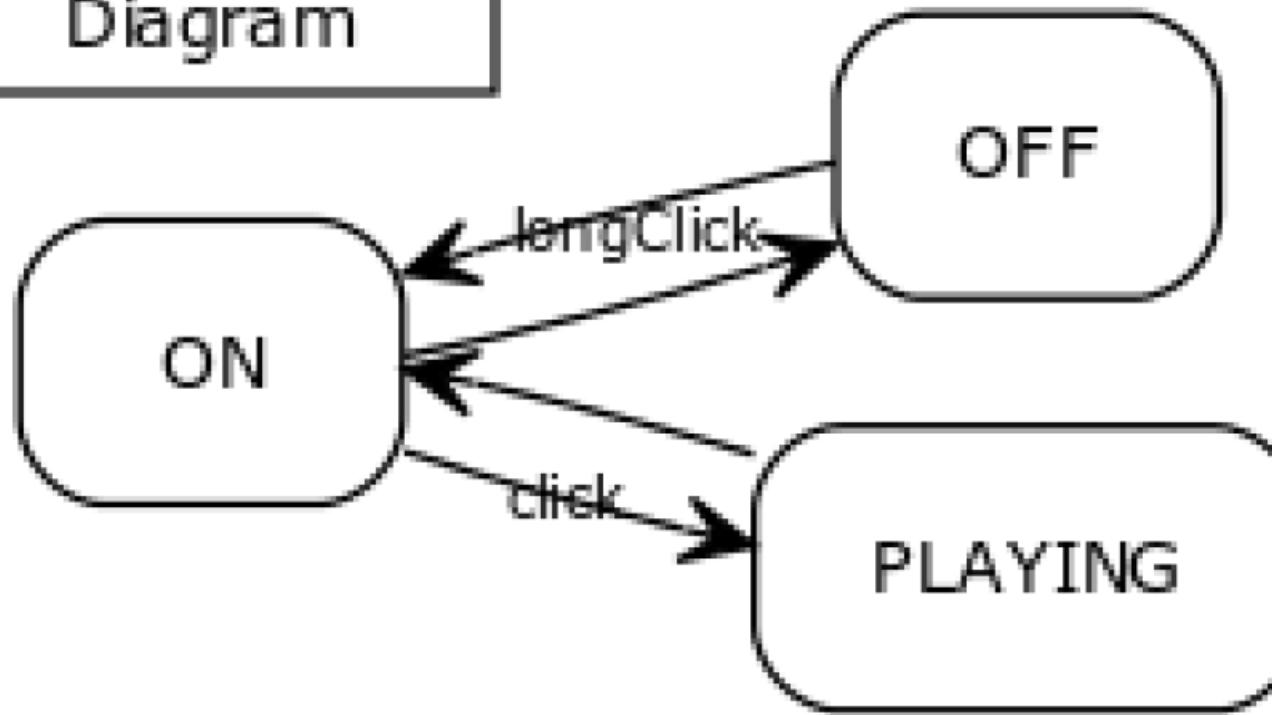


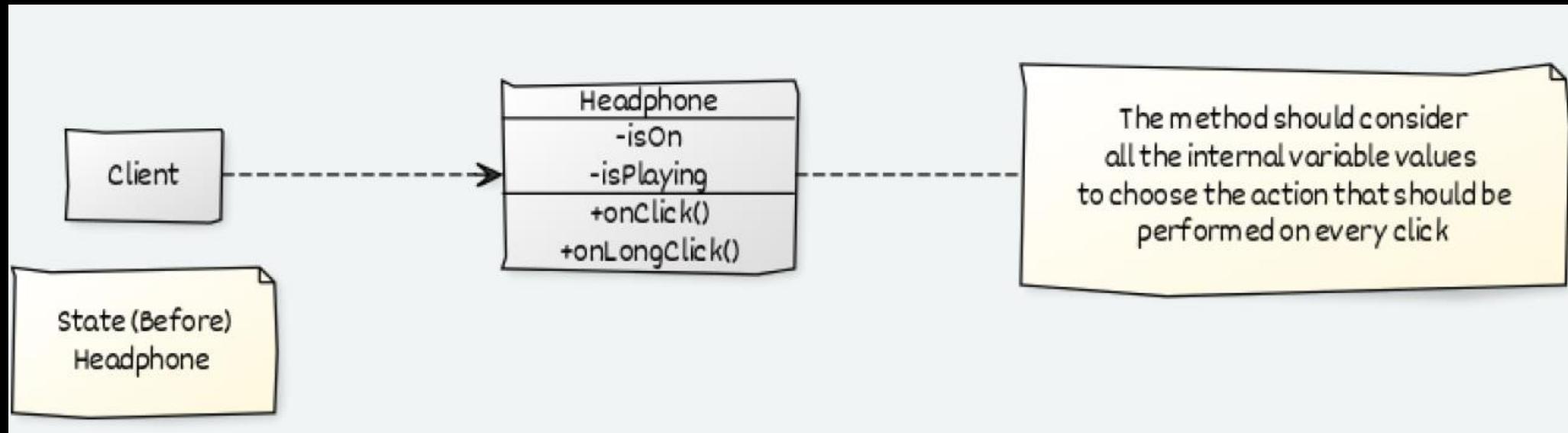
Permitir a um objeto alterar o seu comportamento quanto o seu estado interno mudar. O objeto irá aparentar mudar de classe.

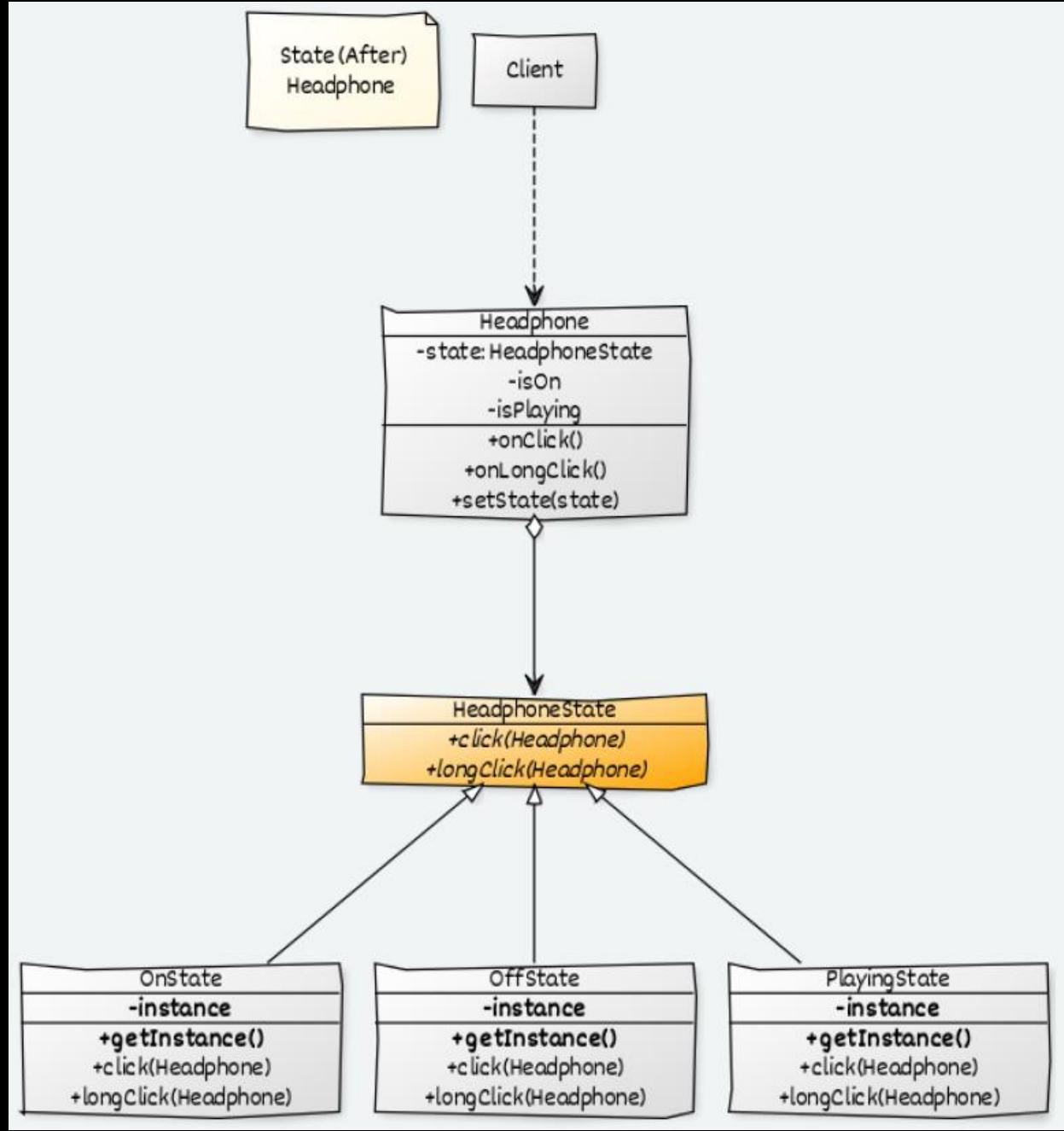
GOF

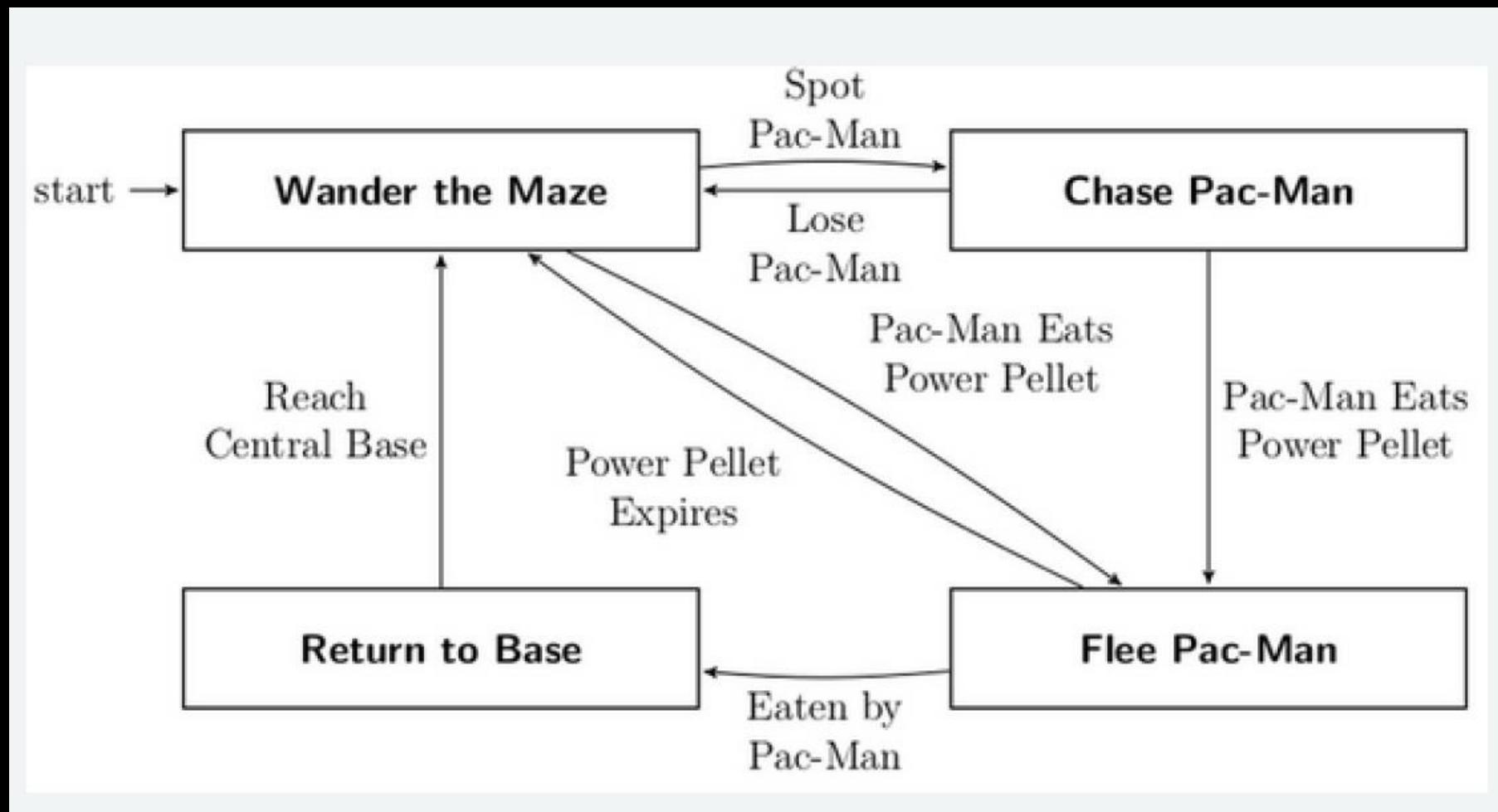


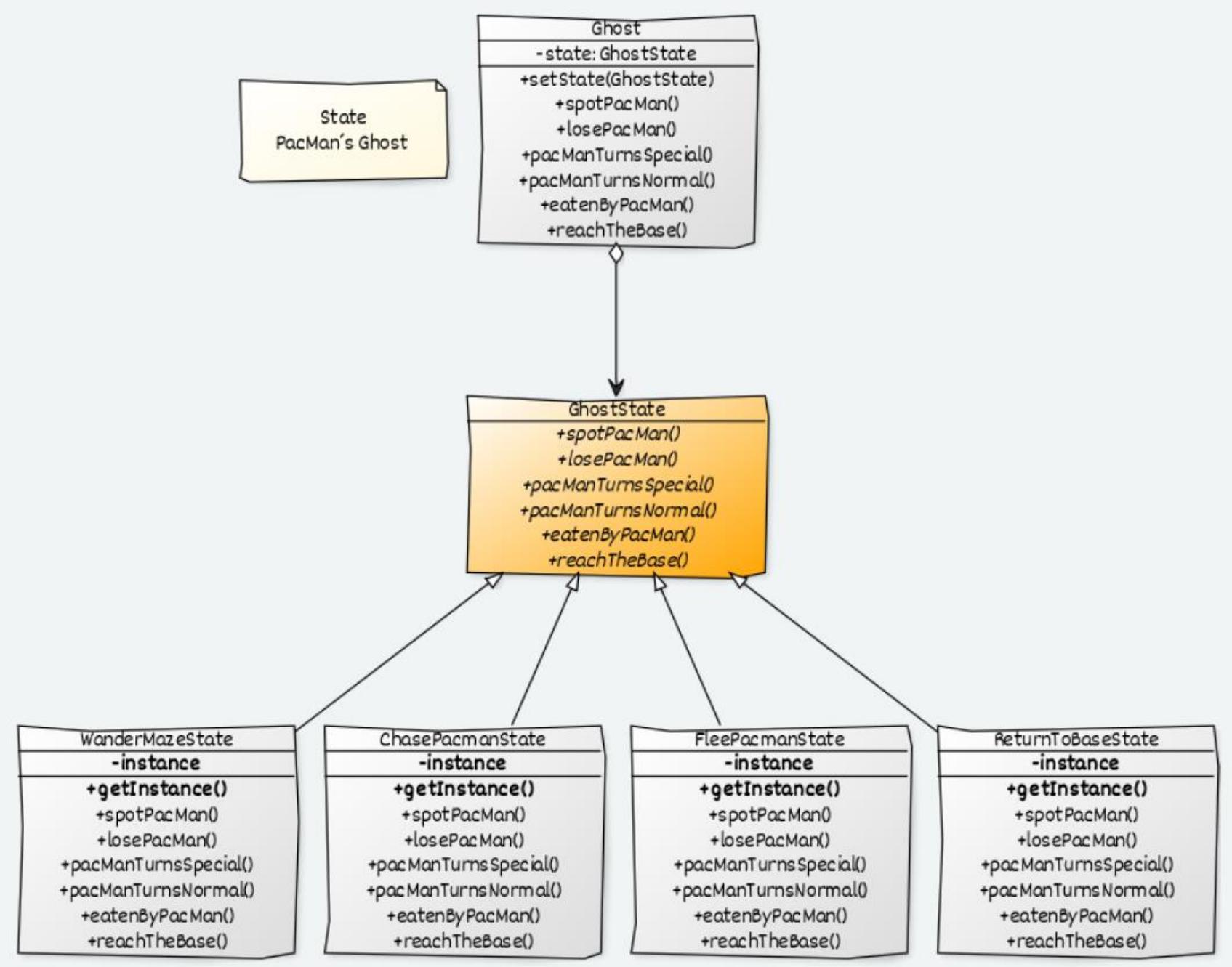
## Headphone Activity Diagram











# STRATEGY

DO GREGO STRATEGY



# PROBLEMAS

- COMO UMA CLASSE PODE UTILIZAR UM ALGORITMO DEFINIDO DINAMICAMENTE?
- COMO POSSO SELECIONAR E TROCAR UMA LÓGICA EM TEMPO DE EXECUÇÃO?



# SOLUÇÃO

- ENCAPSULAR OS ALGORITMOS POSSÍVEIS  
PARA O MESMO PROBLEMA EM UMA  
ESTRUTURA DE STRATEGY
- O CLIENTE IRÁ DELEGAR A EXECUÇÃO  
PARA ESTAS ESTRATÉGIAS AO INVÉS DE  
POSSUIR TODA A LÓGICA INTERNAMENTE

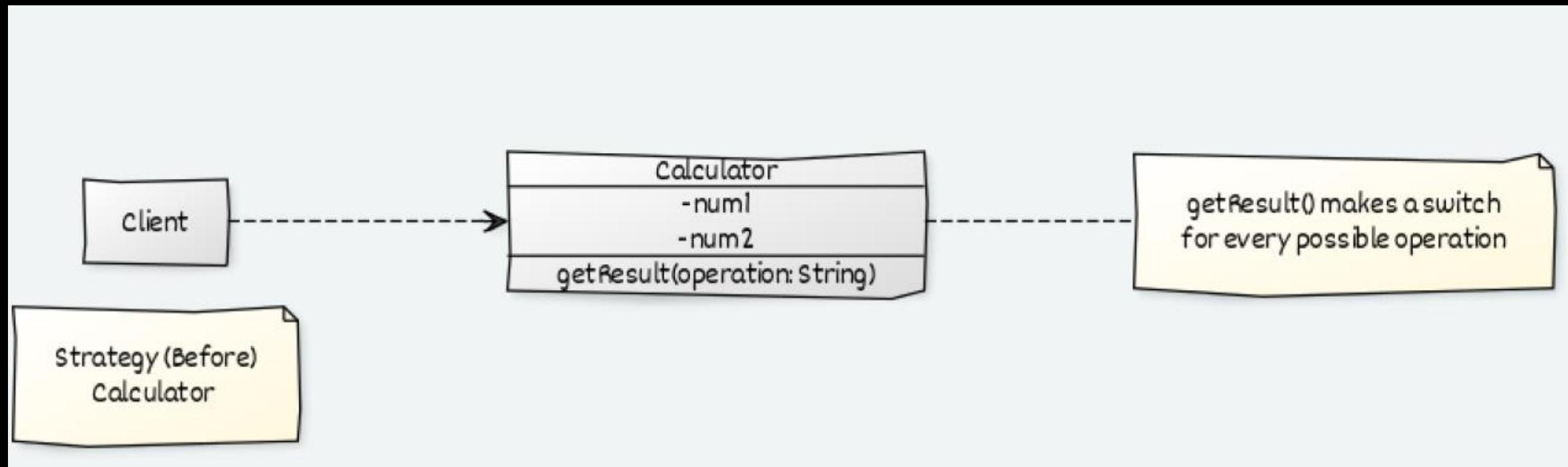


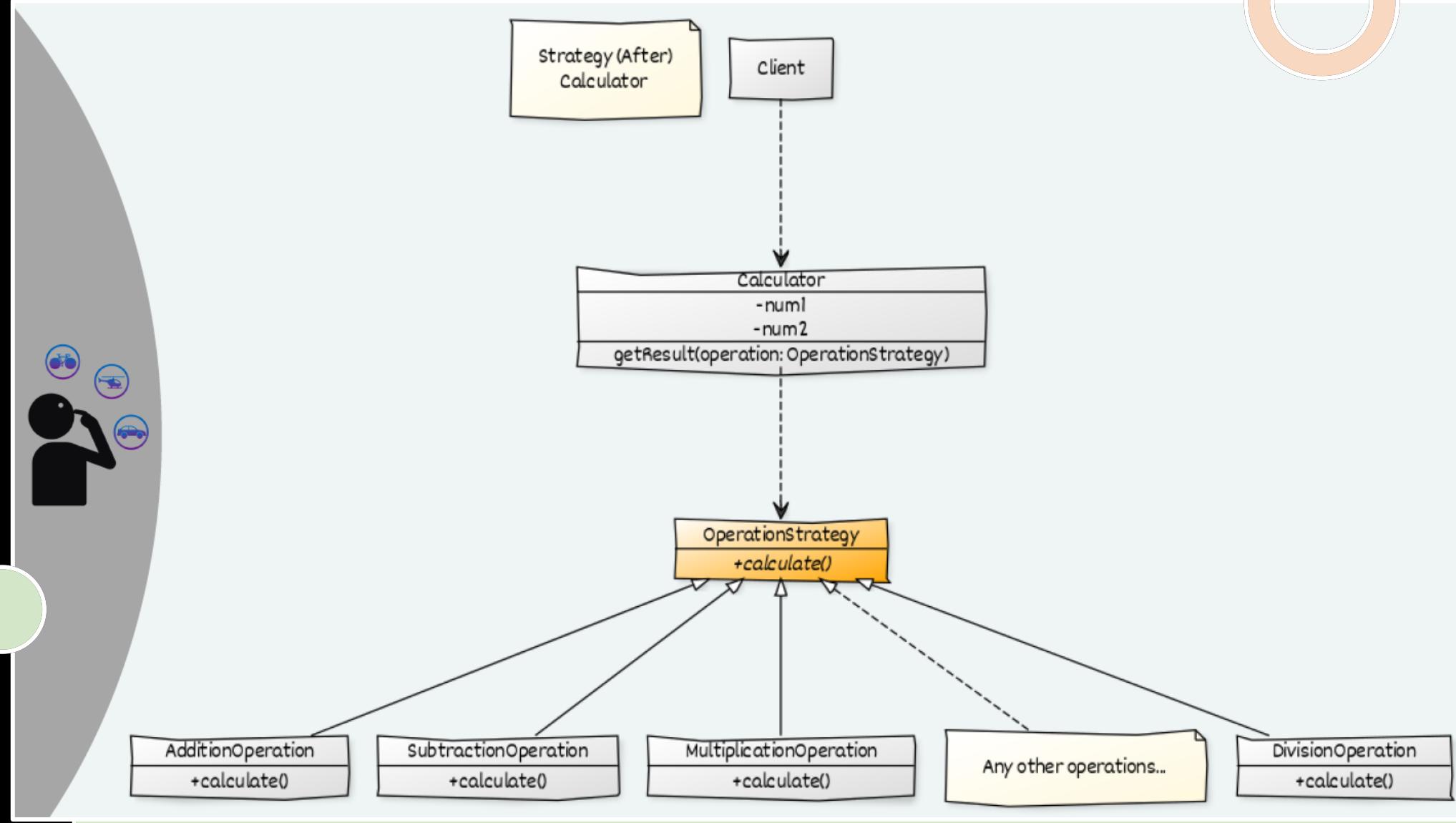


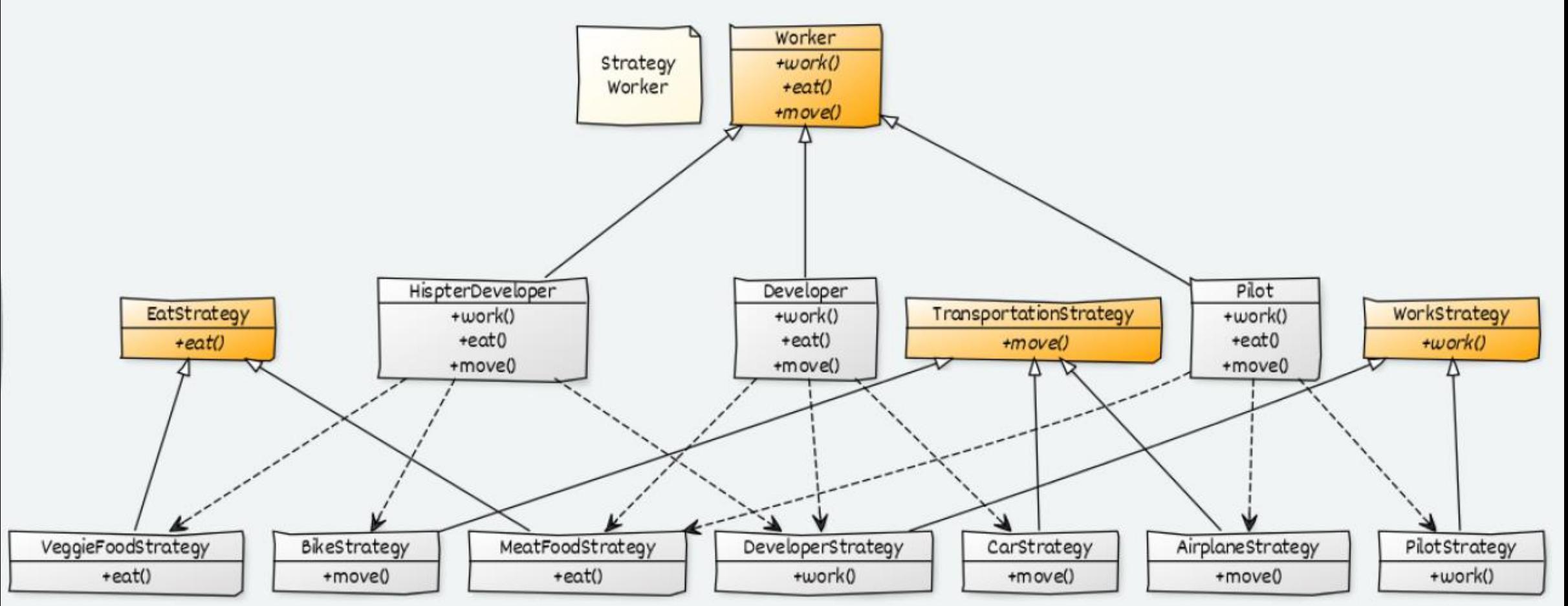
Definir uma família de algoritmos, encapsular cada um, e fazê-los intercambiáveis. Strategy permite que algoritmos mudem independentemente entre clientes que os utilizam.

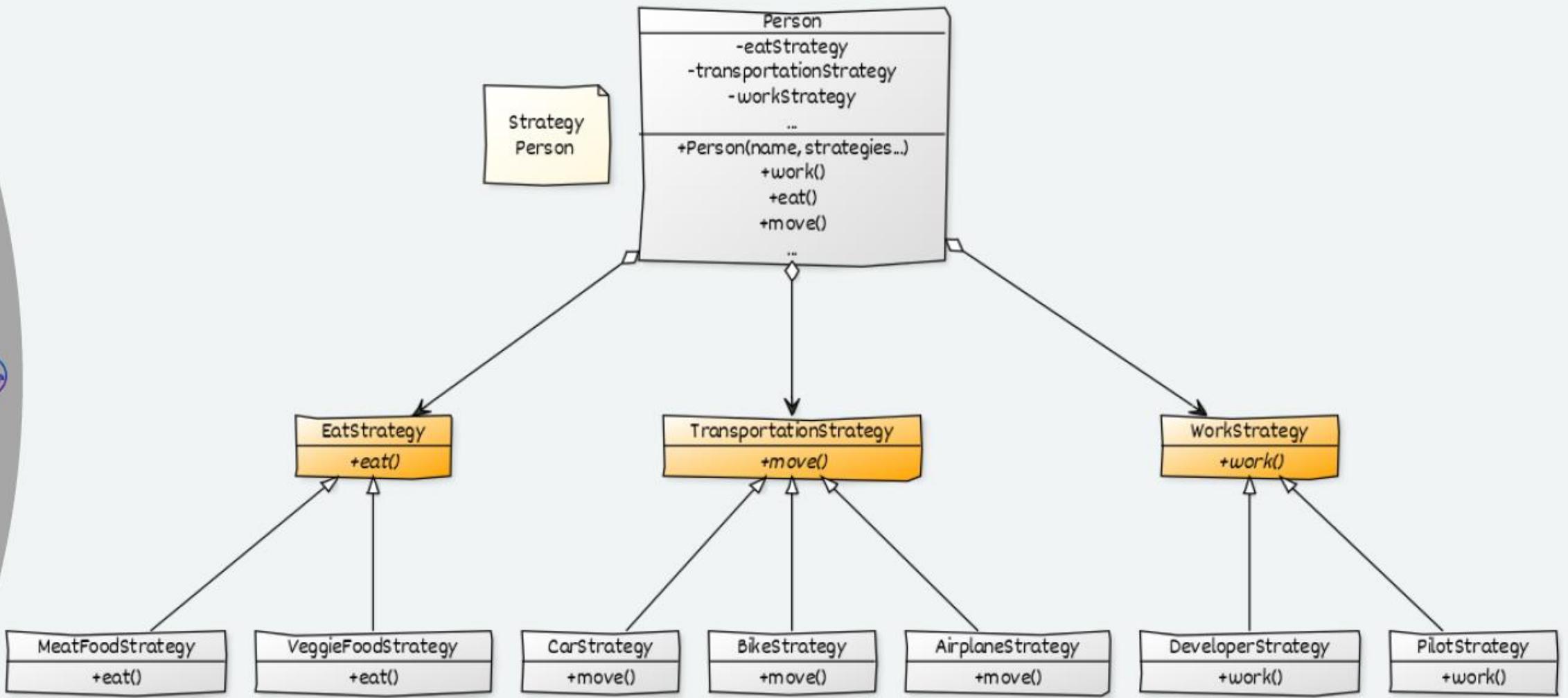


GOF









# TEMPLATE METHOD

NÃO SEI SE PERCEBEU, MAS ISSO AQUI É UM  
TEMPLATE...



# PROBLEMAS

- COMO POSSO UNIR PARTES DE UM CÓDIGO QUE NÃO VARIAM COM PARTES VARIÁVEIS?
- COMO POSSO ALTERAR CERTOS PONTOS DO CÓDIGO MANTENDO UMA ESTRUTURA GERAL?



# SOLUÇÃO

- DEFINIR UMA ABSTRAÇÃO COM TODAS OS PONTOS QUE PODEM SER VARIADOS
- CRIAR UMA TEMPLATE QUE CONTENHA AS PARTES FIXAS E POSSUA PONTOS DE CHAMADAS PARA AS PARTES VARIÁVEIS





Definir o esqueleto de um algoritmo dentro de uma operação, deixando alguns passos a serem preenchidos pelas subclasses. Template Method permite que suas subclasses redefinam certos passos de um algoritmo sem mudar sua estrutura.

GOF



